AD-A260 087

UMENTATION PAGE

Form Approved
OMB No. 0704-0188

(2)

2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
1992	Final, June 1, 1990-October 31, 1992

4. TITLE AND SUBTITLE

Center for Shape Optimization and Material Layout

5. FUNDING NUMBERS

61102F 2304/A1

AFOSR-90-0268A

6. AUTHOR(S)

Konstantin A. Lurie
James I. Northrup

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609

8. PERFORMING ORGANIZATION REPORT NUMBER

AFOSR-TR-93-0099

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DOD/Air Force AFOSR/NM
Bolling AFB DC 20332-6448

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFOSR-90-0268A

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release;
distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The final report reflects the development of a direct method of solving problems of optimal design of systems described by elliptic equations of the 2nd and 4th order. This method is based on a special "polysaddle" transformation of the integrand of the corresponding max min – problem applied to construct the appropriate two–sided bounds; an analytical and computational technique is developed to make this construction possible. Geometrical aspects of the above transform are discussed in detail as well as the material implementation of the corresponding layouts. The results obtained provide a theoretical basis for a direct computational assembling of the global layout from special laminar microstructures determined analytically by the aforementioned technique.

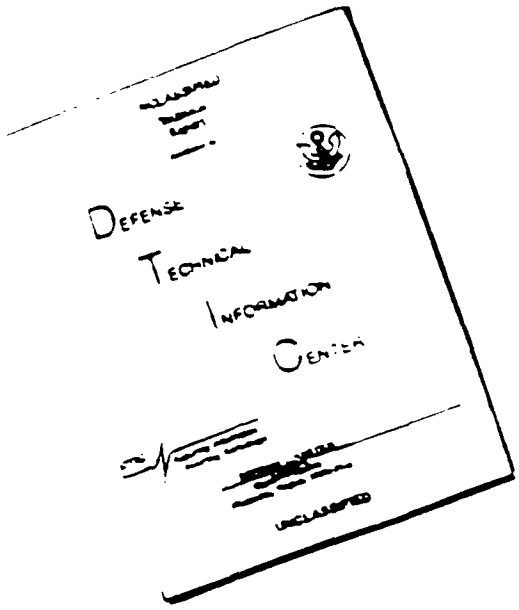
14. SUBJECT TERMS

Direct relaxation, polysaddle transform, convex hulls

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Table of Contents

Statement of Work

The main objective of this work is the development of a unified mathematical technique allowing one to handle optimal design problems *directly*, i.e. without reference to G–closures. The direct approach has originally been put forth in [1], its detailed description given in [2]. This approach has been applied here to problems of the second order arising in the optimal design of heat conducting bodies [3,4], and to the 4th order problems arising in the optimal design of plates [5]. The ultimate aim of this work is to create an exact *analytic* procedure able to provide special microstructures that eventually participate in the optimal layout for non–self–adjoint problems. Currently, these microstructures are worked out *numerically* [6], this step forming an inner loop in the major computational procedure aimed to determine the desired optimal layout. This latter approach (i) provides with only suboptimal sets of composites, and (ii) it has not yet been adjusted to non–self–adjoint problems of design.

The approach reported here does not include such an inner loop; it addresses the computational work at the very final stage, namely, at that of assembling the overall layout from the special composites introduced analytically at the earlier stages of analysis. This analysis is rigorous and provides with genuinely optimal composites.

1 Analytical Aspects

1.1 The Second Order Problems

Direct Solution of an Optimal Layout Problem for Isotropic Heat Conductors in Three Dimensions

Consider the system of equations:

$$q = -\mathscr{D} \cdot \nabla T, \quad \nabla \cdot q = 0 \tag{1}$$

describing the distribution of temperature $T = T(x,y,z)$ within the domain V in $\mathbb{R}^3$ filled by the heat conductor characterized by the heat conductance tensor $\mathscr{D} = \mathscr{D}(x,y,z)$. This one is allowed to be equal either to $\mathscr{D}_+ = u_+ E$ or to $\mathscr{D}_- = u_- E$ where u_+ and u_- are positive constants $(u_+ > u_- > 0)$, and E is a unit tensor $E = ii + jj + kk$:

$$\mathscr{D} = \mathscr{D}(x,y,z) = \chi_1(x,y,z)\,\mathscr{D}_+ + \chi_2(x,y,z)\,\mathscr{D}_-; \tag{2}$$

Here $\chi_{1,2}(x,y,z)$ $(\chi_1 + \chi_2 = 1)$ denote the characteristic functions of regions occupied respectively by $\mathscr{D}_+$– and $\mathscr{D}_-$–materials.

Assume that the temperature T is prescribed along the boundary ∂V of the body:

$$T\Big|_{\partial V} = f \tag{3}$$

and consider the (cost) functional

$$I(T) = -\int_V [T(x,y,z) - T_0(x,y,z)]^2 \, dxdydz \tag{4}$$

where $T_0(x,y,z) \in L_2(V)$ is a known function. We desire to find the characteristic function $\chi_1(x,y,z)$ maximizing the functional (4).

A similar problem in two dimensions has been discussed in [1,2].

To apply a direct method [1,2], we first establish the equivalence of

$$\sup_{\chi_1} I \tag{5}$$

subjected to constraints (1) and (3), and

$$\sup_{\chi_1, T} \inf_{\lambda} J \tag{6}$$

where

$$J = I(T) + \int_V \nabla\lambda \cdot \mathscr{D} \cdot \nabla T \, dxdydz \tag{7}$$

and the "conjugate variable" λ is subjected to

$$\lambda\Big|_{\partial V} = 0 \tag{8}$$

whereas T is subjected to (3).

This latter problem is handled with the aid of two–sided estimates that constitute the core of the direct approach [3].

We start with the following upper bound:

$$\sup_{\chi_1, T} \inf_{\lambda} J = \sup_{T} \sup_{\chi_1} \inf_{\lambda} J \leq \sup_{T} \inf_{\lambda} \sup_{\chi_1} J =$$

$$= \sup_{T} \inf_{\lambda} [I(T) + \int_V G(\nabla T, \nabla \lambda) dxdydz] \tag{9}$$

where (we accept the notation $\xi = \nabla T$, $\eta = \nabla \lambda$)

$$G(\xi,\eta) = \begin{cases} u_+ \xi \cdot \eta \text{ if } \xi \cdot \eta \geq 0, \\ u_- \xi \cdot \eta \text{ if } \xi \cdot \eta \leq 0. \end{cases} \tag{10}$$

We thus calculated $\sup_{\chi_1} J$ explicitly. The upper bound (9) is not final. The function $G(\xi,\eta)$ is not a saddle: it is in fact convex with respect to each argument (but not with respect to their union). For this reason we cannot guarantee the existence of solution of the problem

$$\sup_{T} \inf_{\lambda} [I(T) + \int_V G(\nabla T, \nabla \lambda) dxdydz] \tag{11}$$

on the basis of a saddle–point theorem. But because the arguments ξ,η of $G(\xi,\eta)$ are gradients, we need not demand that the integrand should be a saddle to ensure existence: it is sufficient for this one to be a quasisaddle function [1,2]. We introduce the polysaddle envelope $G^{**}(\xi,\eta)$ of $G(\xi,\eta)$ defined as [1,2]

$$G^{**}(\xi,\eta) = \sup_{\omega} \sup_{b} \inf_{a} \{a \cdot \xi + b \cdot \eta + \omega \cdot \xi \times \eta - \inf_{\xi} \sup_{\eta} [a \cdot \xi + b \cdot \eta + \tag{12}$$

$$+ \omega \cdot \xi \times \eta - G(\xi,\eta)]\};$$

$G^{**}(\xi,\eta)$ thus appears to be an envelope of $G(\xi,\eta)$ built with the aid of linear functions as well as null–Lagrangians $(\xi\times\eta)_1 = \xi_2\eta_3 - \xi_3\eta_2, (\xi\times\eta)_2 = \xi_3\eta_1 - \xi_1\eta_3, (\xi\times\eta)_3 = \xi_1\eta_2 - \xi_2\eta_1$. If $G(\xi,\eta)$ is convex in η and arbitrary in ξ (which is the case here (see [1,2])), then

$$G^{**}(\xi,\eta) \geq G(\xi,\eta). \tag{13}$$

This property makes it possible to use $G^{**}(\xi,\eta)$ instead of $G(\xi,\eta)$ in (11) and thus arrive at the new upper bound of the functional (6).

To compute (12), we first apply the operation $\sup_{\eta}$. With the notation

$$H(\xi,\eta) = -\omega\cdot\xi\times\eta + G(\xi,\eta)$$

we obtain

$$h(\xi,b) = \sup_{\eta}[b\cdot\eta - H(\xi,\eta)] = \begin{cases} 0 \quad \text{if} \quad b + \omega\times\xi - u\xi = 0, \quad u_{-} \leq u \leq u_{+} \\ +\infty \ \text{otherwise.} \end{cases} \tag{14}$$

The next step is given by the operation

$$\inf_{a}\{a\cdot\xi - \inf_{\xi}[a\cdot\xi - (-h(\xi,b))]\}. \tag{15}$$

which yields the concave envelope of $-h(\xi,b)$ with respect to the ξ–variable for fixed b.

This envelope is found to be

$$\inf_{a}\{a\cdot\xi - \inf_{\xi}[a\cdot\xi - (-h(\xi,b))]\} = \begin{cases} 0, \quad \zeta \in \Xi, \\ -\infty \quad \xi \notin \Xi \end{cases} \tag{16}$$

where Ξ is the convex hull of the arc $u \in [u_-,u_+]$ of the curve γ in the ξ–space

$$\gamma\colon\ b = -\omega\times\xi + u\xi = S(u)\cdot\xi, \quad u \in [u_-,u_+], \tag{17}$$

or, explicitly in terms of ξ,

$$\xi = S^{-1}(u)\cdot b, \quad u \in [u_-, u_+],$$

$$S^{-1}(u) = \frac{u}{u^2+\omega^2}E + \frac{\omega\omega}{u(u^2+\omega^2)} - \frac{\omega\cdot\epsilon}{u^2+\omega^2}$$

where $\epsilon = -E\times E$ denotes the Levi–Civita tensor of the third rank.

Note that the above mentioned arc rests on the points $\xi_+ = S^{-1}(u_+)\cdot b$ and $\xi_- = S^{-1}(u_-)\cdot b$; this arc is defined by (14) as the set of points in ξ–space for which $h(\xi,b) < \infty$.

We should now describe the convex hull of this set. An extensive analysis of related computational aspects is given in Section 2; here we give but a general sketch of this geometric object. This one turns out to be the convex body in R^3 ξ–space bounded by a surface $\partial\Xi$ comprised of two sheets. One of these sheets is a developable surface produced by straight lines connecting the point ξ_+ with other points of the arc (17); another sheet emerges from the same construction, this time associated with the point ξ_- . The two sheets intersect along the arc (17) and along the chord connecting the vertices ξ_+ and ξ_-. This chord is given by

$$\xi = tS^{-1}(u_+)\cdot b + (1-t)S^{-1}(u_-)\cdot b, \quad t \in [0,1]. \tag{18}$$

In view of the operation $\sup_{b}$ to be computed next in accordance with the construction (12), we must interpret (15) as the function of argument b for some fixed value of ξ. It is remarkable that this interpretation is quite similar to (16), i.e.

$$\inf_{a}\{a\cdot\xi-\inf_{\xi}[a\cdot\xi-(-h(\xi,b))]\} = \begin{cases} 0, & b\in\mathscr{B}, \\ -\infty, & b\notin\mathscr{B} \end{cases} \tag{19}$$

where $\mathscr{B}$ is a convex body in the R^3 b–space bounded by a surface $\partial\mathscr{B}$ of the same type as the boundary $\partial\Xi$ of Ξ. This surface is also combined of two developable sheets, this time produced by straight lines connecting the vertices $b_+ = S(u_+)\cdot\xi$ and $b_- = S(u_-)\cdot\xi$ with points of the curvilinear arc (18) (this one becomes *a curvilinear arc* in b–space!) On the other hand, the curvilinear arc (17) in ξ–space is interpreted in the b–space as *a chord* connecting the points b_+ and b_-.

The operation $\sup_{b}$ participating in (12) is now reduced to

$$\sup_{b\in\mathscr{B}} b\cdot\eta\ . \tag{20}$$

Because the chord (17) and the arc (18) belong to the boundary $\partial\mathscr{B}$ of $\mathscr{B}$, they should be tested for optimality in terms of the operating (20). Since the body $\mathscr{B}$ is convex and its boundary $\partial\mathscr{B}$ is assembled from two developable sheets, its tangent planes participating in the computation of supremum will touch $\partial\mathscr{B}$ either at its vertices b_+ and b_- or along the arc (18). All these possibilities are embodied in the operation

$$\sup_{b\in[b_+,b_-]} b\cdot\eta \tag{21}$$

where $[b_+,b_-]$ means the *closed* arc (18).

The analysis of this operation together with the subsequent operation $\sup_{\omega}$ shows that the supremum

$$\sup_{\omega} \sup_{b\in[b_+,b_-]} (b\cdot\eta + \omega\cdot\xi\times\eta)$$

is defined by the formula

$$\sup_{\omega} \sup_{b\in[b_+,b_-]} (b\cdot\eta+\omega\cdot\xi\times\eta) = \begin{cases} u_+ \cos 2\chi, & 0 \le \tan\chi \le \sqrt{\dfrac{u_-}{u_+}} \\ (u_+ + u_-)\cos^2\chi - \sqrt{u_+ u_-}\sin^2\chi, & \sqrt{\dfrac{u_-}{u_+}} \le \tan\chi \le \sqrt{\dfrac{u_+}{u_-}} \\ u_- \cos 2\chi, & \sqrt{\dfrac{u_+}{u_-}} \le \tan\chi \le \infty . \end{cases} \qquad (22)$$

Here, 2χ denotes the angle between vectors ξ and η. The expression (22) realizes the upper bound $G^{**}(\xi,\eta)$ for the function $G(\xi,\eta)$ defined by (10). The first and third lines in (22) can be achieved as we apply the pure $\mathscr{D}_+$– and $\mathscr{D}_-$–materials respectively. The second line can be achieved by a rank 1 laminate. To show this, we introduce at each point the plane spanned by vectors ξ,η. Then we introduce a layered composite assembled from $\mathscr{D}_+$ and $\mathscr{D}_-$ –materials with layers directed perpendicularly to the ξ,η–plane and bisecting the angle 2χ between ξ and η. For this microstructure, with its effective tensor $\mathscr{D}_0$ and the concentration t chosen in accordance with the rule

$$[tu_- + (1-t)u_+]^2 = u_+ u_- \tan^2\chi, \qquad (23)$$

the expression for $\xi\cdot\mathscr{D}_0\cdot\eta$ coincides with that in the second line of (22). Observe that the requirement $0 \le t \le 1$ combined with (23) gives birth to inequalities

$$\sqrt{\frac{u_-}{u_+}} \le \tan\chi \le \sqrt{\frac{u_+}{u_-}}$$

involved in (22).

The attainability of the upper bound

$$\sup_{T}\inf_{\lambda} J \le \sup_{T}\inf_{\lambda}\left[I(T) + \int_V G^{**}(\xi,\eta)dxdydz\right]$$

if thereby proved. In this argument, we may consider special microstructures applied above as *admissible layouts* generating the *lower bound* for the same functional $\sup_{\chi,T}\inf_{\lambda} J$. Both bounds are shown to be the same, and the existence of optimal layout is thus demonstrated. Observe that this layout is *locally two–dimensional* since at each point (x,y,z) we have two vectors ξ,η that define a plane; in this plane we obtain the layout essentially the same as that described in [1,2].

This overall layout is thus to be constructed from pure $\mathscr{D}_+$ and $\mathscr{D}_-$ – materials and also from rank one laminate assembled from them; the rule (22) shows how this construction should be arranged.

Direct Solution of an Optimal Layout Problem for Isotropic and Anisotropic Heat Conductors on a Plane

This problem differs from the previous one only in the definition of a set U of admissible controls (materials). Now, instead of two isotropic materials $\mathscr{D}_+ = u_+E$ and $\mathscr{D}_- = u_-E$, we assume that U involves the isotropic material $\mathscr{D}_1 = uE$ and the anisotropic material $\mathscr{D}_2 = R\mathscr{D}_{20}R^T$, $R = e_1'e_1 + e_2'e_2$, $R^T = e_1e_1' + e_2e_2'$, $\mathscr{D}_{20} = d_1e_1e_1 + d_2e_2e_2$, $0 <$

$u < d_1 < d_2$. This anisotropic material is allowed to take various orientations at each point; particularly, the polycrystals are allowed to emerge.

The overall analysis of this problem is given in [4] (see attachment). The optimal layout is assembled from the pure $\mathscr{D}_1$–material, the (properly oriented) $\mathscr{D}_{20}$–material, and the rank one laminate assembled from them. A rule is indicated showing the ranges through which either of these three materials should be applied.

These examples illustrate the direct technique of [1,2] applied to the second order problem of heat theory.

The following example is referred to the fourth order problem arising in the theory of plates.

1.2 The Fourth Order Problems

Direct Approach in the Optimal Design of Plates.

The state of equilibrium of a thin plate is described by the equation

$$\nabla\nabla \cdot\cdot\ \mathscr{D} \cdot\cdot\ \nabla\nabla w = q, \qquad \nabla = i\frac{\partial}{\partial x} + j\frac{\partial}{\partial y}, \tag{24}$$

$$(x, y) \in \Sigma$$

where w denotes the normal displacement, $\mathscr{D}$ the tensor of stiffness, and q the transverse load density. The boundary $\partial\Sigma$ of the plate will be assumed clamped, this property requiring the boundary conditions

$$w\Big|_{\partial\Sigma} = \frac{\partial w}{\partial n}\Big|_{\partial\Sigma} = 0. \tag{25}$$

The tensor $\mathscr{D} = \mathscr{D}(x,y)$ will be allowed to take at each point (x,y) one of the two admissible values $\mathscr{D}_1$, and $\mathscr{D}_2$ associated with two available constructive materials:

$$\mathscr{D}_i = k_i a_1 a_1 + \mu_i (a_2 a_2 + a_3 a_3), \quad i = 1,2. \tag{26}$$

Here and below, a_1, a_2, a_3 represent an orthonormal basis in the space of 2nd rank symmetric 2×2–tensors, i.e.

$$a_1 = \frac{1}{\sqrt{2}}(ii + jj), \quad a_2 = \frac{1}{\sqrt{2}}(ii - jj), \quad a_3 = \frac{1}{\sqrt{2}}(ij + ji). \tag{27}$$

Introducing the characteristic functions $\chi_1(x,y)$, $\chi_2(x,y)(\chi_1+\chi_2 = 1)$ of domains occupied by materials $\mathscr{D}_1$ and $\mathscr{D}_2$ respectively, we by analogy with (2) may write

$$\mathscr{D}(x,y) = \chi_1(x,y)\,\mathscr{D}_1 + \chi_2(x,y)\,\mathscr{D}_2 . \tag{28}$$

The function $\chi_1(x,y)$ will be sought to maximize some weakly continuous functional $I(w)$ of solution to the problem (24), (25). As typical example, we may take the functional

$$I(w) = -\int_{\Sigma} [w(x,y) - w_0(x,y)]^2 dxdy, \quad w_0(x,y) \in L_2(\Sigma) \tag{29}$$

To apply the direct approach to this problem, we go through the same steps as before. The problem

$$\sup_{\chi_1} I$$

subjected to (24), (25) is equivalent to

$$\sup_{\chi_1, w} \inf_{\lambda} J \tag{30}$$

where

$$J = I(w) + \int_{\Sigma} (\nabla\nabla\lambda \cdot\cdot \mathscr{D} \cdot\cdot \nabla\nabla w - \lambda q)dxdy \tag{31}$$

and λ is subjected to

$$\lambda\Big|_{\partial\Sigma} = \frac{\partial\lambda}{\partial n}\Big|_{\partial\Sigma} = 0\,. \tag{32}$$

To construct an upper estimate for (30), we proceed as in (9) and obtain

$$\sup_{w}\inf_{\lambda} J = \sup_{w}\sup_{\chi_1}\inf_{\lambda} J \le \sup_{w}\inf_{\lambda}[I(w) - \int_{\Sigma} \lambda q \, dxdy +$$

$$+ \int_{\Sigma} G(\nabla\nabla w, \nabla\nabla\lambda)dxdy] \tag{33}$$

where (we accept the notation $\xi = \nabla\nabla w$, $\eta = \nabla\nabla\lambda$)

$$G(\xi,\eta) = \begin{cases} \xi \cdot\cdot \mathscr{D}_1 \cdot\cdot \eta, \text{ if } \xi \cdot\cdot \mathscr{D}_1 \cdot\cdot \eta \ge \xi \cdot\cdot \mathscr{D}_2 \cdot\cdot \eta, \\ \\ \xi \cdot\cdot \mathscr{D}_2 \cdot\cdot \eta, \text{ if } \xi \cdot\cdot \mathscr{D}_1 \cdot\cdot \eta \le \xi \cdot\cdot \mathscr{D}_2 \cdot\cdot \eta. \end{cases} \tag{34}$$

The bound (33) is still to be improved with the aid of the polysaddle transform similar to (12). We go from $G(\xi,\eta)$ to a new integrand $G^{**}(\xi,\eta)$ computed by the formula

$$G^{**}(\xi,\eta) = \sup_{\omega,d}\sup_{b}\inf_{a}\{a\cdot\cdot\xi+b\cdot\cdot\eta+\omega\cdot\cdot(\xi\times\eta)+d\xi\cdot\cdot T\cdot\cdot\eta -$$

$$-\inf_{\xi}\sup_{\eta}[a\cdot\cdot\xi+b\cdot\cdot\eta+\omega\cdot\cdot(\xi\times\eta)+d\xi\cdot\cdot T\cdot\cdot\eta - G(\xi,\eta)]\}. \qquad (35)$$

Apart from null–Lagrangians $(\xi\times\eta)_1 = \xi_2\eta_3 - \xi_3\eta_2,\ldots$, similar to those involved in the 2nd order case, this formula includes the null–Lagrangian $\xi\cdot\cdot T\cdot\cdot\eta$ with the 4th rank tensor T defined as

$$T = a_1a_1 - a_2a_2 - a_3a_3\,. \qquad (36)$$

This null–Lagrangian is specific for the 4th order operator appearing in (24) namely, the following identity holds

$$\nabla\nabla\cdot\cdot\,T\,\cdot\cdot\,\nabla\nabla w \equiv 0.$$

The transform (35) possesses properties similar to those of (12); particularly, if $G(\xi,\eta)$ is convex in η and arbitrary in ξ (which is the case here), then (eq. (12))

$$G^{**}(\xi,\eta) \geq G(\xi,\eta)\,,$$

and $G^{**}(\xi,\eta)$ can be used to improve the bound (33).

We now should compute $G^{**}(\xi,\eta)$. This time we cannot characterize the boundary $\partial\mathcal{B}$ that now appears in a manner similar to that in Section 1.1 simply in terms of two developable sheets; the geometry of this surface is more complicated. Nevertheless, the general scheme of Section 1.1 works well, and we are able to carry out the analysis of corresponding necessary conditions. Detailed calculations are exposed in [5] (see attachment); the complete classification of ranges has been provided for the case when tensors ξ and η are coaxial. The

ultimate layout involves in this case laminates of rank one and matrix laminates of rank two, their layers being oriented along the (common) main axes of tensors ξ and η.

The method works well also in the general situation when tensors ξ and η are arbitrary. The corresponding calculations have been initiated, and they show that laminates of rank one appear as microstructures that are optimal within certain ranges of ξ and η. For other ranges, general (non–matrix) laminates of rank two apply; the final classification of ranges is still to be completed.

2 Computational Aspects

2.1 Introduction

In solving the materials layout problem, we wish to evaluate the function $G^{**}(\xi,\eta)$. This function is constructed from the function $G(\xi,\eta)$ as follows.

$$G^{**}(\xi,\eta) = \sup_{D,\omega}\sup_{b}\inf_{a}\{a\cdot\xi + b\cdot\eta + D\xi T\eta + \omega\cdot\xi\times\eta$$
$$-\inf_{\zeta}\sup_{\theta}[a\cdot\zeta + b\cdot\theta + D\zeta T\theta + \omega\cdot\zeta\times\theta - G(\zeta,\theta)]\}$$

where the terms in this function are defined as follows. The arguments ξ and η are symmetric tensors of size two. If, however, we choose a basis for such tensors we can think of ξ and η as being three-tuples (elements of $\Re^3$) so that operations such as $\xi\times\eta$ make sense.[1] In particular, we choose the basis

$$\alpha_1 = \frac{1}{\sqrt{2}}(ii+jj) \qquad \alpha_2 = \frac{1}{\sqrt{2}}(ii-jj) \qquad \alpha_3 = \frac{1}{\sqrt{2}}(ij+ji)$$

We express ξ and η in this basis as $\xi_i = \xi\cdot\cdot\alpha_i$ and $\eta_i = \eta\cdot\cdot\alpha_i$ for $i = 1,2,3$.[2] The dummy variables a, b, ω, ζ and θ are also thought of as being in $\Re^3$.[3] D is a scalar. T is defined to be the matrix representation of the operation $\mathcal{T} = \alpha_1\alpha_1 - \alpha_2\alpha_2 - \alpha_3\alpha_3$ in the basis $\{\alpha_1,\alpha_2,\alpha_3\}$. In other words,

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

In order to define $G(\xi,\eta)$ recall that k_-, k_+, μ_- and μ_+ are all scalar quantities. Define

$$K_+ = k_+\xi_1\eta_1 + \mu_+(\xi_2\eta_2 + \xi_3\eta_3) \quad \text{and}$$

[1] That is, $\xi\times\eta = \begin{vmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \xi_1 & \xi_2 & \xi_3 \\ \eta_1 & \eta_2 & \eta_3 \end{vmatrix}$ where the set $\{\alpha_1,\alpha_2,\alpha_3\}$ is the basis of the space of tensors.

[2] In other words, think of the tensor ξ as being a two-by-two matrix with elements $\xi_{(k,l)}$. Then the scalar $\xi_i = \xi\cdot\cdot\alpha_i = \sum_{k,l}\xi_{(k,l)}(\alpha_i)_{(k,l)}$ and $\xi = \sum_i \xi_i\alpha_i$.

[3] Note the analogous roles played by ξ and ζ, and by η and θ. We will subsequently be considering the space $\Re^6 = \Re^3\times\Re^3$ with ξ and ζ belonging to the same $\Re^3$ subspace and η and θ belonging to the same $\Re^3$ subspace. Later, b will also be thought of as belonging to the same space as η and θ.

$$K_- = k_-\xi_1\eta_1 - \mu_-(\xi_2\eta_2 - \xi_3\eta_3)$$

Then finally, $G(\xi,\eta) = \max\{K_+, K_-\}$.

With $G^{**}(\xi,\eta)$ now defined, we consider how to evaluate it. We first turn our attention to the terms underlined below.

$$G^{**}(\xi,\eta) = \sup_{D,\omega}\sup_{b}\inf_{a}\{a\cdot\xi + b\cdot\eta + D\xi T\eta + \omega\cdot\xi\times\eta$$
$$-\inf_{\zeta}\underline{\sup_{\theta}}[a\cdot\zeta + \underline{b\cdot\theta + D\zeta T\theta + \omega\cdot\zeta\times\theta - G(\zeta,\theta)}]\}$$

We will call this $\overline{s}(\zeta,b)$; that is,

$$\overline{s}(\zeta,b) = \sup_{\theta}[b\cdot\theta + D\zeta T\theta + \omega\cdot\zeta\times\theta - G(\zeta,\theta)]$$

We determine analytically that

$$\overline{s}(\zeta,b) = \begin{cases} 0 & \text{at all } \zeta \text{ such that} \\ & b + DT\zeta - [ka_1a_1 + \mu(a_2a_2 + a_3a_3)]\zeta + \omega\times\zeta = 0 \\ \infty & \text{at any other } \zeta \end{cases}$$

where $k = k_+ - \delta(k_+ - k_-)$ and $\mu = \mu_+ - \delta(\mu_+ - \mu_-)$ and where the parameter δ is allowed to range through the interval (0,1). (So, in other words, k ranges through all of the values between k_- and k_+, and μ ranges through all of the values between μ_- and μ_+.) Note that the equation

$$b + DT\zeta - [ka_1a_1 + \mu(a_2a_2 + a_3a_3)]\zeta + \omega\times\zeta = 0$$

can be thought of as the parametric representation of a surface in $\Re^6$, with δ acting as the parameter, and $\{\zeta_1,\zeta_2,\zeta_3,b_1,b_2,b_3\}$ being the basis vectors of $\Re^6$. We call this surface $M(\zeta,b)$, so that

$$M(\zeta,b) = b + DT\zeta - [ka_1a_1 + \mu(a_2a_2 + a_3a_3)]\zeta + \omega\times\zeta = 0$$

It will be using it extensively in subsequent computations.

The second step in evaluating $G^{**}(\xi,\eta)$ is to consider also the terms double-underlined below.

$$G^{**}(\xi,\eta) = \sup_{D,\omega}\sup_{b}\underline{\underline{\inf_{a}}}\{\underline{\underline{a\cdot\xi}} + b\cdot\eta + D\xi T\eta + \omega\cdot\xi\times\eta$$
$$\underline{\underline{-\inf_{\zeta}\sup_{\theta}}}[\underline{\underline{a\cdot\zeta}} + \underline{b\cdot\theta + D\zeta T\theta + \omega\cdot\zeta\times\theta - G(\zeta,\theta)}]\}$$

We call this $\underline{c}(\xi, b)$; that is,

$$\underline{c}(\xi, b) = \inf_{a}\{a \cdot \xi - \inf_{\zeta}[a \cdot \zeta + \overline{3}(\zeta, b)]\}$$

Now, for each fixed b, we can think of the set of all ξ for which this infimum is attained as being the convex hull of $M(\zeta, b)$. In other words, the set of all ξ for which this infimum is attained is the convex hull of the set of all ζ for which $\overline{3}(\zeta, b)$ is equal to zero, where b is held fixed and where ζ and ξ are thought of as being in the same space, $\Re^3$.

To place this procedure in perspective, consider the surface $M(\zeta, b)$ as existing in $\Re^6$. By fixing b, we take a slice through M in the ζ-direction. The result is some surface in three-dimensional ζ-space. We then convexify this surface; the result is the set $\{\xi \in \Re^3 \mid \underline{c}(\xi, b) \text{ attains its infimum}\}$.

If we were to consider the above procedure taking place for all b, we would describe a six-dimensional body $\mathcal{B}$ which contains $M(\zeta, b)$ as a subset. We imagine slicing through $\mathcal{B}$ in the b-direction by fixing some particular value of ζ, this particular value being ξ. For the sake of brevity, we will refer to this slice through $\mathcal{B}$ in the b-direction as $\mathcal{B}_\xi$ (since ξ is fixed), and we will refer to a slice through $\mathcal{B}$ in the ξ-direction (that is, with b fixed) as $\mathcal{B}_b$. Note that with this notation, $\mathcal{B}_b = \{\xi \in \Re^3 \mid \underline{c}(\xi, b) \text{ attains its infimum}\}$. Figures 1 through 7 provide a schematic representation of these bodies. Note that in these schematics, we have labeled the horizontal axis with a ξ rather than with a ζ; since they are both in the same $\Re^3$ space we could have labeled the axis with either symbol, but we chose ξ since that is the particular value along that axis which we will ultimately be interested in. Note also that b and ξ are drawn as if they were scalar quantities, when in fact they represent three-tuples.

As a third step in the evaluation of $G^{**}(\xi, \eta)$, we consider the triple-underlined terms below.

$$G^{**}(\xi, \eta) = \sup_{D,\omega} \underline{\underline{\underline{\sup_{b}\inf_{a}}}}\{\underline{\underline{\underline{a \cdot \xi}}} + \underline{\underline{\underline{b \cdot \eta}}} + D\xi T\eta + \omega \cdot \xi \times \eta$$

$$\underline{\underline{\underline{-\inf_{\zeta}\sup_{\theta}}}}[\underline{\underline{\underline{a \cdot \zeta}}} + \underline{\underline{\underline{b \cdot \theta}}} + \underline{D\zeta T\theta + \omega \cdot \zeta \times \theta - G(\zeta, \theta)}]\}$$

Computing this latest supremum amounts to evaluating $\sup_b b \cdot \eta$ where b is allowed to range throughout $\mathcal{B}_\xi$ for any fixed ξ. So in other words, given ξ and η, we first find the surface $M(\zeta, b)$. Then, for each fixed b we slice

through that surface in the ζ-direction, and compute the convex hull of the resulting surface. Doing this for each b value obtains for us the body $\mathcal{B}$. Now with the ξ we have been given acting as a particular value of ζ, we slice through $\mathcal{B}$ in the b-direction to obtain $\mathcal{B}_\xi$. We then compute

$$\sup_{b \in \mathcal{B}_\xi} b \cdot \eta$$

We call this quantity $\bar{b}_{D,\omega}(\xi, \eta)$. This process is demonstrated schematically in Figures 8 and 9.

The fourth and final step in evaluating $G^{**}(\xi, \eta)$ is straightforward. We compute

$$\sup_{D,\omega} \{\bar{b}_{D,\omega}(\xi, \eta) + D\xi T\eta + \omega \cdot \xi \times \eta\}$$

By far the most difficult step is computing $\bar{b}_{D,\omega}(\xi, \eta)$. This, in turn, is only made difficult by the cumbersome definition of $\mathcal{B}_\xi$ as the ξ-slice of the union of all convex hulls of b-slices of $M(\zeta, b)$. The difficulty arises not in computing the ξ-slice itself, but in computing all possible b-slices of $M(\zeta, b)$. Clearly, we cannot expect to compute an infinitude of complex hulls numerically! We will discuss possible solutions to this problem in Section 4.

2.2 Representations of $M(\zeta, b)$

Recall that

$$M(\xi, b) = b + DT\xi - [ka_1a_1 + \mu(a_2a_2 + a_3a_3)]\xi\omega \times \xi$$

We call this the *tensor formulation* of $M(\xi, b)$. We could instead write this as a linear system of three equations in three unknowns. To do this, we must first express ω in the basis $\{\alpha_1, \alpha_2, \alpha_3\}$, so that $\omega_i = \omega \cdot\cdot \alpha_i$ for $i = 1, 2, 3$. For the sake of brevity, we write $A = \omega_1$, $B = \omega_2$ and $C = \omega_3$. The system then becomes

$$\begin{aligned}
\{D - [k_+ - \delta(k_+ - k_-)]\}\xi_1 - C\xi_2 + B\xi_3 &= -b_1 \\
C\xi_1 + \{-D - [\mu_+ - \delta(\mu_+ - \mu_-)]\}\xi_2 - A\xi_3 &= -b_2 \\
-B\xi_1 + A\xi_2 + \{-D - [\mu_+ - \delta(\mu_+ - \mu_-)]\}\xi_3 &= -b_3
\end{aligned}$$

We call this the *linear system formulation*[4] of $M(\xi, b)$. We can now write this system in a "matrix × vector = vector" fashion as $(\mathcal{A}+\delta\mathcal{D})\vec{\xi} = \vec{b}$ where $\mathcal{A}$ is a full matrix and $\mathcal{D}$ is diagonal.[5] We call this the *matrix formulation* of $M(\xi, b)$. This notation essentially divorces the computation from its physical context, but is a more standard formulation from the point of view of numerical analysis.

So as a first step in numerically evaluating the function $G^{**}(\xi, \eta)$ we suppose that b is fixed and compute a discretized version of the surface $M(\xi, b)$. Let N be the number of points in the discretization; we will write the discretized version of the $M(\xi, b)$ function as $M_N(\vec{\xi}, \vec{b})$, or more simply as M_N, and we will refer to this as the *discrete formulation* of $M(\xi, b)$. M_N is computed by choosing N values of the parameter δ in the interval [0,1] and solving the above linear system for $\vec{\xi}$ at each of these N values of δ. We will refer to these discrete values of δ as $\{\delta^{(i)}\}_{i=1}^{N}$. Likewise, the N discrete values of values of $\vec{\xi}$ will be denoted $\{\vec{\xi}^{(i)}\}_{i=1}^{N}$. Figures 10-14 provide a computer-graphics representation of M_N and its hull for the following test problem.[6]

$$\begin{aligned}
\{2 - [1.5 - \delta(1.5 - 0.1)]\}\xi_1 - 0\xi_2 + 0\xi_1 &= -1 \\
0\xi_1 + \{-2 - [30 - \delta(30 - 2)]\}\xi_2 - 30\xi_2 &= -85 \\
0\xi_1 + 30\xi_2 + \{-2 - [30 - \delta(30 - 2)]\}\xi_3 &= -50
\end{aligned}$$

The points on the discretized surface were computed in FORTRAN and stored in a data file for eventual use in computing the convex hull of M_N. The FORTRAN code used is provided as an appendix. Note that values were assigned to the scalars b, A, B, C and D arbitrarily for the purposes of these computations.

2.3 Computing the convex hull

At this point, we have a numerical approximation to a slice in the b-direction, for some fixed b, through the six-dimensional body $\mathcal{B}$. We now need to

[4]The scalar $B = \omega_2$, above, should not be confused with the set $\mathcal{B}$ from the previous section.

[5]ξ is now written as $\vec{\xi}$ to emphasize the fact that the current formulation is divorced from the physical (tensor) meaning of ξ. That is, $\vec{\xi}$ will henceforth always be thought of as a three-tuple rather than as a symmetric tensor.

[6]Illustrations are also provided for the *hull* test surface $x = (1-t)\sin(16t)$, $y = (1-t)\cos(16t)$, $z = \frac{1}{2} + 8(t - \frac{1}{2})^3$, where t ranges between 0 and 1. See Figures 15-18.

compute a numerical approximation to its convex hull.[7] Just as the discrete analog of the surface $M(\xi, b)$ is a set of N points, the discrete analog to the surface of the convex hull will be a set of M triangular facets, $\{\triangle_j\}_{j=1}^{M}$. The algorithm we used for computing the discretized convex hull is called the "Giftwrapping Algorithm" and is a standard algorithm in computational geometry. We describe it, and provide our C language implementation of the algorithm, in an appendix. Algorithms which are theoretically more efficient are known, but are difficult to implement. At this stage of the project, we chose the sufficiently efficient Giftwrapping Algorithm for its (relative) ease of implementation. Also, at this stage of the project, it is just as important to be able to view a computer graphics representation of the hull as it is to actually compute the hull. For this reason, our C program also displays the convex hull by making use of the X Windows graphics and interface library, xlib. Figure 13 provide a computer-graphics representation of the convex hull. (Note that while the program makes use of shading to provide a "three dimensional look" to the hull on the computer screen, printouts of the screen are necessarily black-and-white and so lack the look of depth.)

Being able to view the convex hull is important for three reasons. First of all, its the only practical method for determining that there is no error in the convex hull computation. More importantly, if we consider the image of the convex hull in the context of its original physical derivation, certain facets of the hull represent various classes of laminar composites. For example, points on the body which lie on a line segment connecting two points on the surface represent rank-one laminates. Finally, observing the convex hull for some problems provides insight for methods of attacking the problem analytically rather than numerically. For example, developable surfaces[8] on the convex hull might indicate linearity in the original problem which could be exploited.

2.4 Computing $\sup_{b \in \mathcal{B}_\xi} b \cdot \eta$

2.4.1 Pattern Search Approach

As we previously described the procedure for evaluating $G^{\bullet\bullet}(\xi, \eta)$, we would have to compute the above convex hull *for every possible value of* b in order

[7] That is, we need to approximate the set $\{\xi \in \Re^3 \mid \underline{\varphi}(\xi, b) \text{ attains its infimum}\}$.

[8] That is, surfaces which can be traced out by the motion of a line segment through space. The length of the line segment is allowed to vary as the segment moves.

then to compute the body $\mathcal{B}$. Clearly, this is impractical! And yet, without $\mathcal{B}$, we cannot then slice through $\mathcal{B}$ in the b-direction so as to compute $\mathcal{B}_\xi$ at the given ξ. One way around this dilemma is to realize that we are not, in fact, interested in computing either $\mathcal{B}$ or $\mathcal{B}_\xi$ at all; we are interested only in computing $\sup_{b \in \mathcal{B}_\xi} b \cdot \eta$. This is an optimization problem with a linear objective function and a nonlinear constraint set. In fact, the constraint set is not only nonlinear, its also non-smooth, and even non-convex. Worst of all, we do not even have an analytic description of its boundary. Rather, for any given b value we can answer the question "is b in $\mathcal{B}_\xi$?"[9] but we have no other information about the constraint set for nearby values of b or ξ. In particular, we cannot even determine if we are on the boundary of the set except by sampling nearby points and repeating the convex hull computations at each of these points. Thus, even obtaining approximate derivative information about the boundary is impossible. Therefore, we need to use a no-derivative method of optimization, such as a pattern search method, in order to solve the problem. We are currently at this stage of computer program development in the project.

2.4.2 Higher Dimensional Approach

Another approach to alleviating this lack of an easily-defined constraint set would be to consider the full problem in its six dimensional setting rather than approaching it as a sequence of three dimensional problems, as we propose above. In this approach, we would compute a quasi-convex hull of $M(\xi, b)$ as in Figure 6. While algorithms for computing convex hulls abound in the Computational Geometry literature, we know of no algorithms for computing quasi-convex hulls. It appears, however, that an easy generalization of the Giftwrapping Algorithm, which we used to compute our three dimensional convex hulls, might enable us to compute directly the quasi-convex hull of the six dimensional body. We are currently developing such an algorithm, but have not yet implemented it.

[9] We would answer this question by fixing b, computing the convex hull of $M(\xi, b)$, then fixing ξ, and seeing if that ξ values lie inside the convex hull we computed. Note that numerical error can be a very sensitive issue here; ξ's just barely inside the boundary of the convex hull might be computed to be outside the boundary either because of roundoff error in the coordinates, or due to discretization error caused by numerically approximating the convex hull.

2.5 Appendices to Computational Aspects

The following appendices are intended primarily for those involved in using or maintaining the *hull* software.

2.5.1 Appendix A - Algorithm Used to Compute M_N

For our materials layout problem, both ξ and b are three dimensional. So the overall space is six dimensional. The ξ-surface is not a discrete set of points, but rather a trajectory in three dimensions. Its convex hull is a body in three space. By varying b, we can build up a six dimensional figure: the $\mathcal{B}$ envelope.

In a ξ-space setting for our full problem, the surface whose convex hull needs to be computed is given parametrically in the form

$$\begin{aligned}
\{D - [k_+ - \delta(k_+ - k_-)]\}\xi_1 - C\xi_2 + B\xi_1 &= -b_1 \\
C\xi_1 + \{-D - [\mu_+ - \delta(\mu_+ - \mu_-)]\}\xi_2 - A\xi_2 &= -b_2 \\
-B\xi_1 + A\xi_2 + \{-D - [\mu_+ - \delta(\mu_+ - \mu_-)]\}\xi_3 &= -b_3
\end{aligned}$$

which we write as $(\mathcal{A} + \delta\mathcal{D})\vec{\xi} = \vec{b}$ where $\mathcal{A}$ is a full matrix and $\mathcal{D}$ is diagonal. Here, the entries of the matrices $\mathcal{A}$ and $\mathcal{D}$ are known constants. In the overall problem, both $\vec{\xi}$ and $\vec{b}$ are unknown, but as a sub-problem we consider $\vec{b}$ to be fixed. So the algorithm for computing a single convex hull proceeds as follows.

1. Fix an arbitrary three-tuple $\vec{b}$.
2. Let $\{\delta^{(i)}\}_{i=1}^{N}$ be n evenly spaced values of δ in the interval $[0, 1]$.
3. For each δ_i, solve the equation $(\mathcal{A} + \mathcal{D}\delta)\vec{\xi} = \vec{b}$ for $\vec{\xi}$, where $\mathcal{A}$ is a full matrix and $\mathcal{D}$ is diagonal.
4. This gives us a set of vectors $\{\vec{\xi}^{(i)}\}_{i=1}^{N}$ which form the discretized surface in ξ-space.
5. From this, we compute a set of triangles $\{\triangle_j\}_{j=1}^{M}$ which is the discrete analog of the convex hull. We used the Giftwrapping Algorithm to perform this computation.

2.5.2 Appendix B - Notes on a Proposed Pattern Search

At this point we have a method for building up the set $\mathcal{B}$. (The method is theoretical, not practical.) We now turn our attention to computing $\sup b \cdot \eta$ efficiently. Notice from Figure 19 that for a given ξ and a given η we do not need to know all of set $\mathcal{B}$. In Figure 19, for instance, all we need to know is the top boundary of $\mathcal{B}$ for some fixed ξ. In order to determine how this fact might be used, we consider Figure 20. Here we imagine a two dimensional b-space and we draw the b-image of $\mathcal{B}$ for some fixed ξ. (That is, we draw $\mathcal{B}_\xi$. In Figure 20, $\mathcal{B}_\xi$ is shaped like a lemon slice.) For this particular example, we have the unit direction vector η cutting through $\mathcal{B}_\xi$, but this is not necessarily the case in general; η might not cut through $\mathcal{B}_\xi$.

First, it is illustrative to consider the possible correspondences between features of Figure 19 and features of Figure 20.

- The b-axis in Figure 20 corresponds to the the entire b_1, b_2-plane in Figure 20. Note that for this reason, a single point in Figure 20 might simultaneously correspond to any number of points in Figure 20.

- The line segment $\overline{st}$ in Figure 20 corresponds to the "lemon slice" between the points u and v in Figure 20. In both cases, this set is a b image of $\mathcal{B}$ corresponding to some fixed ξ. That is, it is the the set $\mathcal{B}_{xi}$.

- The point s in Figure 20 corresponds to the point w in Figure 20. This point is the point at which $\sup_{b \in \mathcal{B}_\xi} b \cdot \eta$ is attained.

- The point t in Figure 20 corresponds to the line segment $\overline{uv}$ in Figure 20. In each case, this is the set formed by intersecting the b-space with the original surface M. Note that in Figure 20 we draw this as a line segment, rather than as an arbitrary surface, because in the sample problem discussed above, b depends linearly on the parameter δ for fixed ξ.

- The point t in Figure 20 corresponds to the point u in Figure 20. This is the point at which $\sup_{b \in M|_\xi} b \cdot \eta$ is attained. That is, it maximizes $b \cdot \eta$ over the original surface, with ξ fixed, rather than over the entire set $\mathcal{B}_\xi$.

It is very important to keep in mind that we do not actually know the location of the horizontal line which passes through the point s in Figure 20. We could

only find this line by computing an infinite number of convex hulls. Even approximating its location would require an impractically large number of convex hull computations, since b is really a three-tuple and not a scalar. If we did know the location of the horizontal line, and therefore the location of s, our problem would be easily solved.

We now consider how these observations can be used to write a constructive algorithm for determining the point w (or s). Again, it is useful to consider Figure 20 in this discussion. The first algorithm we consider is clearly impractical, however it illustrates some ideas we can use in a more realistic algorithm. Recall that ξ and η are given as input, since ultimately we are trying to evaluate $G^{**}(\xi, \eta)$. Also, note that $M(\xi, b)$ is known; in this case, it is the surface given parametrically by $(\mathcal{A} + \mathcal{D}\delta)\vec{\xi} = \vec{b}$.

1. Optimize $b \cdot \eta$ over $M(\xi, b)$ with the ξ given. That is let $C = \sup_{\delta \in [0,1]} \eta(\mathcal{A} + \delta\mathcal{D})\xi$. In Figure 20, this value of C would be attained at the point t. We now wish to somehow move "upwards" from point t towards the better solution at point s.

2. Randomly choose a value for b. There are three possibilities.

   (a) This new value of b might not be relevant to computing a new value of C. For example, consider the points b' and b'' in Figure 20. In the case of the point b', the corresponding ξ-space (horizontal line) would not intersect the surface M at all, and so has no bearing on finding the point s. In the case of the point b'', the corresponding ξ space does intersect M, but not in the vicinity of the b-space setting corresponding to our fixed ξ. Since these b values are not of interest to us, we return to step 2 and randomly guess a new value of b.

   (b) This new value of b might be relevant, but might not increase the value of C. For example, consider the point b_t in Figure 20. For this b value, $b \cdot \eta$ is not increased. So, we return to step 2 and randomly guess a new value of b.

   (c) The new value of b might increase the value of C. For example, choose a point in Figure 20 which is slightly above the point b_t. For this new b value, $b \cdot \eta$ is increased. Let C be equal to this new value of the dot product, and return to step 2, to look for a still better b.

Clearly, this strategy of randomly guessing b values is not practical, especially since b is actually a three-tuple and not a scalar. The above algorithm does show how the merit of a b value can be judged. It is left, then, to replace the random guessing with a better search scheme, such as a pattern search. For example, given a value of b, we would consider 6 points within ϵ of b, in the positive and negative directions of each of the three coordinate axes. The value of the objective function would be computed at any of these six points which turn out to be in the set $\mathcal{B}_\xi$. The point with the lowest value is taken as the starting point in the next step of the iteration.

2.5.3 Appendix C - Introduction to the the *hull* program

This appendix explains how the 3D convex hulls are rendered by the program *hull*, how to use the program, and provides file listing for the code.

Running the program Hull runs in a Unix and X Windows environment, such as is commonly found on computer workstations. At the shell prompt, type "hull" to invoke the program. *Hull* will read in a number of (x, y, z) coordinate triples from a file called *surface.data*, so it is assumed that this file resides in the same subdirectory as the *hull* program. The data is stored as floating point numbers, three per line, separated by blank spaces.

The main screen Most of the screen should now be taken up by the image of a surface, super-imposed on the coordinate axes. (See Figures 3 and 4.) In the upper left will be a dark rectangle labeled "Quit". Clicking inside this rectangle with any mouse button will exit from the program. Technically, these dark rectangles are not "widgets" in the sense of X-Windows terminology, so we will refer to them as gadgets.

To the right of the "Quit" gadget is the "Save hull" gadget. Clicking in this gadget will save the hull data into the file *hull.data*. Each line of the file contains nine numbers representing one triangle: the first three numbers are the (x, y, z) coordinates of the first vertex, the second set of three numbers is the coordinates of the second vertex, and so on. The numbers are separated by blank spaces.

On a color monitor, there will be a color bar to the right of the "Save hull" button. The color bar displays 66 colors and 66 shades of grey. The colors used are the 66 standard "named" colors in X-Windows. The shades

of grey are the ones used to paint the convex hull. On a black-and-white monitor, the color bars will not be drawn.

The overhead-view gadgets To the right, and slightly below the color bar, will be a large rectangle surrounded by gadgets. The large rectangle is an overhead map of the display area. In the center of the map is the origin. Emanating downward from the origin is the x-axis. The y-axis extends to the right. The small square which surrounds the origin represents the extent of the x and y axes as they are displayed on the screen. The letter "e" represents the location of the observer's eye. The letter "s" represents the target of the observers eye (the "see point"). The letter "l" represents the location of the lamp, or light source.

Between the eye point and the see point is an imaginary plane called the "view plane". This is the rectangle onto which the 3D image is projected, for purposes of viewing. The letter "v" in the overhead map is always between the "e" and the "s" — it represents the location of the view plane.

To the right of the overhead map is an "altimeter" which shows the z-coordinate of the eye point, the see point, and the light source. The dot in the middle of the altimeter represents the height of the origin. The vertical bar just inside the altimeter represents the extent of the z-axis as it is displayed on the main screen.

The gadgets marked "Expand" and "Ex" double the size of the map area, so everything inside the map becomes smaller. The gadgets marked "Shrink" and "Sh" cut the size of the map area in half.

Clicking on the "Clockwise" gadget with the first (left-most) mouse button will rotate the eye point five degrees, with the see point being the center of rotation. The rotation will occur in the sense of latitude lines. Notice that the image in the main screen changes, as does the location of the eye point as displayed in the overhead map. Clicking on the same gadget with the second (middle) mouse button will rotate the see point five degrees about the origin. Clicking with the third (right-most) mouse button will rotate the light source about the see point.

The "Counter C" gadget performs counter-clockwise rotations in the same fashion. The "Go over" gadget rotates these points up towards the positive z-axis (in the sense of longitude lines). The "Go under" gadget rotates them downwards.

The "See point" and "Eye point" gadgets move the view plane between these two points. Repeated clicking on the "See point" gadget, for example, will move the view plane closer to the see point. This is indicated by the location of the "v-slider" which connects the two gadgets. Moving the view plane in and out is the best way to control the size of the image.

The "Zoom in" and "Zoom out" gadgets can be used to move the eye point and light point closer or farther from the see point. It can also be used to move the see point closer or farther from the origin. These gadgets are less useful than the "See point" and "Eye point" gadgets, because (for example) zooming the eye point towards the see point also moves the view plane, automatically. The net effect does not actually increase the size of the image by much.

The eye point can also be rotated around the latitude lines by using the left and right arrow keys on the keyboard. The up and down arrow keys rotate along the longitude lines. Again, the center of rotation is the see point.

Finally, notice that clicking into the overhead map or the altimeter can be used to change the locations of these points much more abruptly. Clicking with the first mouse button in the overhead map will, for example, jump the eye point to that location.

Hull gadgets The gadgets below these "overhead map" gadgets control the shape of what is actually seen. Clicking on the "COMPUTE CONVEX HULL" gadget causes the convex hull to actually be computed. If more than 4096 triangles are needed to compute the hull, the program will beep at you and write an error message to the DECterm window which evoked it. Clicking on the "surface" gadget shows only the surface generated by connecting the (x, y, z) coordinate triples from the data file. The "Wireframe" gadget calls up a wireframe representation of the convex hull of this surface. The "White" gadget draws the conves hull in all white, with hidden surfaces eliminated from the image and individual facets of the hull outlined in black. The "Shaded" gadget draws the hull most realistically: hidden surfaces are eliminated and the remaining surfaces are painted shades of grey depending on the angle of incident light emanating from the lamp. The "Highlight" gadget attempts to show off important aspects of the image. The visible portion of the surface is painted yellow (white, on a black and white monitor). If part

of the surface is on the exterior of the hull, but is on the far side of the hull and so not visible, it is painted red. If a part of the surface is interior to the hull,it is painted brown. If the line segement connecting the beginning of the surface to the ending point is also on the convex hull, it is painted green (white, on a black and white monitor). This green line usually represents a sharp corner on the hull.

The "Less fine" and "More fine" gadgets control how many of the data points in the data file are actually used. When the program starts, it attempts to find 16 data points to use. If the data file contains 128 data points, then only every eighth of these is used. Clicking on the "Less fine" gadget cuts the number of points used in half. Clicking on the "More fine" gadget doubles them, up until all of the data points in the file are being used. The purpose of this is to allow the user to manipulate "crude" images in order to get the proper view, before performing the time-consuming calculations associated with more refined images. So for example, if one wanted to rotate the image quickly, it might make sense to click on the "surface" gadget and the "Less fine" gadget first, so that the rotations would appear more quickly on the screen. Note: the program only allows 4096 triangles to be displayed on the screen. Trying to refine the image too much could exceed that limit. If so, the computer will "beep" at you, and only render the first 4096 triangles it has calculated (with three of these being the coordinate axes.)

On some occasions the user might want to see only the trailing portion of the surface, or the beginning portion. The next four gadgets control this. Clicking on the "+ t min" gadget chops off segments of the surface associated with small parameter values. The "- t max" gadget chops off segments of the surface associated with large parameter values.

Scaling gadgets Below the hull gadgets are six gadgets which double of halve the scaling of the x, y and z axes.

Files The various parts of the program as a whole can be divided into three categories. (1) Sections which compute the convex hull. (2) Sections which transform that data into a 3D image. (3) Sections which make use of the X-Windows library to render that image on the workstation screen. These routines are stored in a number of separate files, listed below.

main.c This file contains the main() program. All global varibles are initially declared here. Also, the main drawing routine, refreshWindow(), is found in this file.

2d.c This file contains the functions which transform device-independent coordinates into actual screen coordinates. (In the device-independent coordinate system, the computer display is thought of as a unit square, with the point (0,0) in the lower left, and (1,1) in the upper right.) The device-independent coordinates themselves are generate by functions in the file 3d.c.

3d.c This file contains functions for transforming 3D coordinates to 2D coordinates. The 3D coordinates are called "object space" coordinates, and the 2D coordinates are device-independent. This file also contains the functions which initialize the matrices needed to perform this transformation.

colorx.c The suffix "x" on a file name denotes that it pertains to the X-Windows system. In this case, the functions in this file are used to initialize the colors and the grey-scale used by the program.

draw.c This file contains all of the 2D drawing functions, such as functions used to draw lines and rectangles on the screen.

eventx.c The suffix "x" on a file name denotes that it pertains to the X-Windows system. This file contains the function which waits for various "events" (mouse button clicks, keyboard key presses, etc.) and acts upon them.

gadgetdata.c This file contains the location and string associated with each gadget.

getdata.c This file contains the function which reads the data file. The data file consists of (x, y) coordinate pairs, one pair per line, which define the surface to be convexified. The first line of this data file is an integer which tells the program how many coordinate pairs to expect.

hull.c This file contains the functions which compute the convex hull of the surface.

initx.c The suffix "x" on a file name denotes that it pertains to the X-Windows system. This file contains the functions which first initialize and establish the connection to the X-server.

minmax.c This file contains various functions for computing minimums and maximums of floating point values.

overhead.c This file contains the functions which convert overhead map coordinates into screen coordintes.

quitx.c This file contains the function which cleanly severs the connection to the X-server.

sort.c This file contains the functions needed to perform a quick-sort. This is used to sort the facets of the convex hull and the coordinate axes, so that they are rendered on the screen in the correct order.

textx.c This file contains the function which sets up the X-Windows font to be used.

windowx.c This file contains the function which opens a window on the X-server screen.

theIcon An icon for the application when it is closed. Note that the DecWindows window manager does not use this icon.

global.h A global include file for all of the ".c" files.

prototypes.h A file of the C function prototypes, also included in each of the ".c" files.

makefile The make file for the program.

surface.data This file contains the data for the surface whose hull is to be computed. The first line of this file consists of an integer, the number of points on the surface. Each remaining line of the file contains three floating point numbers, the (x, y, z) coordinates of each point. The floating point numbers are separated by spaces.

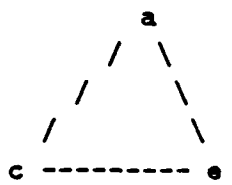
hull.data This file is generated by the hull program. The first line of this file consists of an integer, the number of triangles on the hull. Each remaining line of the file contains nine floating point numbers, the (x, y, z) coordinates of each of the three vertices of a triangle. The floating point numbers are separated by spaces.

surface.f This file contains a typical FORTRAN program for creating the file *curve.data*

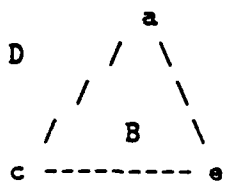
Generating convex hulls with the gift-wrapping algorithm

The general algorithm Suppose we have a set of five points in 3D and we want to find their convex hull.

For the moment, take it as given that we have somehow found three points which we know form one face of the hull. For the sake of visualization, let us set up our coordinate system so that these three points are sitting on the floor.



The other two points are hovering somewhere above these three.



Choose any of the three edges of the face as an edge to "wrap" your gift paper around. For example, choose the edge $\overline{ac}$. Also choose one of the points on this edge for computing a vector which is normal to the face: let us choose a.

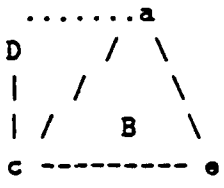
Now compute the unit vector $\vec{n}$, normal to this face.

$$\vec{n} = (c \vec{-} a) \times (e \vec{-} a), \text{ normalized.} \tag{1}$$

We loop through all of the vertices which are not on this face:

```
case vertex = B:
  consider the triangle [a,c,B]
  construct a unit vector, _m_, normal to this triangle
  let rho(B) = _n_ <dot product> _m_
case vertex = D:
  consider the triangle [a,c,D]
  construct a unit vector, _m_, normal to this triangle
  let rho(D) = _n_ <dot product> _m_
```

As we loop through all of these vertices, we look for the smallest ρ value. In other words, we look for the face whose normal is "most obtuse" to the face we already have. So we choose D.



Now we have two faces. How do we procede from here? We keep a list of all of the *edges* we have created, and we make sure that we consider each edge exactly one time, in terms of "wrapping" the gift paper around it. So for example, we have now considered the edge $\overline{ca}$ exactly one time. We should never consider it again. Still on our list of edges are the following:

```
[c,e] created with our initial face
[e,a] created with our initial face
[c,D] created by our second face
[a,D] created by our second face
```

We loop through all of these edges until we have considered exactly once every edge ever construced. Note that when we get around to creating the face [a,D,B] we have to be careful not to put the edge $\overline{aD}$ in our list of edges a second time. (It was already added to our list when we created the face [a,c,D].)

For a more in-depth discussion of this algorithm, see *Computational Geometry, an Introduction* by Franco Preparata and Michael Shamos.

Degenerate cases For our computations, we are required to also consider two dimensional degenerate cases, where all of the points lie on a linear manifold. Then, the convex hull is also a two dimensional object. Applying the Gift Wrapping algorithm now, there is no clear choice of "most obtuse" faces, since they are all in fact parallel to one another. Assuming that the points are considered for candidacy in the same order at every iteration of the algorith, we would find that the same four points are being chosen over and over again, regardless of how many points are in the set.

That is, imagine five coplanar points, *a,b,c,d,e*. In extending face [a,b,c] we could choose either point *d* or point *e*; they are both equally good choices. Our program considers these points in order, so it would choose point *d*. Likewise in extending face [a,b,d] we would choose point *c*, and so on. Point *e* would never be used, and yet point *e* might well lie on the two dimensional hull.

In our implementation of the Gift Wrapping algorithm, we overcome this problem by adding the extra criterion that for two equally good choices of new faces, we choose the one which uses the point least-frequently utilized at that phase of the computation. Thus, in the above example, [a,b,c] would be extended to *d*, but [a,b,d] would be extended to *e*, since *c* has been used more often so far than *e*.

The first face as a special case The way in which we obtain the first face of the hull is a special case.

To find the first vertex in the face choose, say, the lowest point in the set of vertices as our starting vertex. We are guaranteed that this vertex will be on the convex hull.

To find the second vertex in the face run line segments through the first point and each of the other $N-1$ points. Choose, say, the one whose

angle with the x-axis is largest.

To find the final vertex in the face run a plane through the plane through the edge joining those two points and rotate that plane around this new line segment. Compute the normal vector of these planes as they hit each of the other $N-2$ points. Choose, say, the one whose angle with the x-axis is largest.

These three point form the first face of the convex hull.

Rendering 3D objects

3D to 2D transformations Given a set of triangles in three dimensions (the hull), we now consider how these triangles would be projected onto a "view plane" so that their image can be displayed on the computer screen. More precisely, we need to find a transformation which maps points in three dimensions to pixels on the computer screen.

There are four coordinates systems which must be used for this transformation. (See Figure 5.)

1. The coordinate system of the 3D object space. (x = right, y = up, z = inward, away from observer)
2. The coordinate system of the 2D view plane; the origin in this coordinate space is the point c.
3. The device-independent screen coordinate space $[0,1] \times [0,1]$.
4. The device-dependent screen coordinate space (pixels).

There are several important points in these spaces. Note that since there is more than one space, each of these points might be represented using different coordinates. (See Figure 21.)

- the point *eye*, where the observer's eye is
- the point *see*, where the observer is looking (i.e. the target of his view)
- the point *c*, which is midway between *eye* and *see*; note that the scalar variable ρ controls exactly where between *see* and *eye* the point *c* lies
- the point *o*, which is the point in the object space that is to be plotted

- the point v, on the view plane, to which o is mapped; v lies somewhere along the line segment connecting *eye* to o

There are also several mappings being used.

- the mapping PP maps from o to v, but both o and v are represented using their 3D object space coordinates
- the mapping T converts v from its 3D coordinates to its 2D natural view plane coordinates (with origin $= c$)
- the mapping DVI converts v from its 2D view plane coordinates to its 2D screen device-independent coordinates $[0,1] \times [0,1]$
- the mapping DVD converts v from its 2D screen device-indpendent coordinates to device-dependent coordinates (pixels)

How the mappings are derived

PP: represent v as $v = \alpha o + (1-\alpha)eye$, and *note* that $(c-v)\cdot(c-eye) = 0$. Then solve for α. One you have α, you can compute v as a linear combination of o and *eye*.

T: note that $d_2 = P(see + (0,1,0))$ lies in the y-axis (up axis) of the view plane. Therefore the y-axis is given as $t_2 = d_2 - c$. Note that $t_1 = see - c$ is perpendicular to the view plane. Therefore $d_1 = t_1 \times t_2$ must be the x-axis of the view plane. Then normalize d_1 to v_1 and t_2 to v_2. Let v_1 be the first row of a matrix A and v_2 be the second row of A. Then $A(v-c)$ converts a point v in the view plane from its 3D coordinates to its 2D coordinates.

DVI: DVI is device independent. Still, we have to decide how much of the view plane we want the device to capture. That is, the view plane is (of course) infinitely big, and we just want to show some rectangle on that plane. Let $maxV_x$ and $maxV_y$ be the maximimum (and minimum) x and y coordinates that we want to capture. (Without loss of generality we can just let them be 1.) Then we should multiply all view-plane x coordinates by $1/(2maxV_x)$ and y coordinates by $1/(2 * maxV_y)$ so that the rectangle we want to capture is now in the domain $[-.5,+.5] \times$

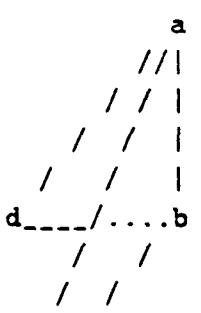
$[-.5, +.5]$. Then we should add .5 to each coordinate Now all the points that are in the rectangle we want to *see* have coordinates in $[0,1] \times [0,1]$.

DVD: *DVD* is device dependent. Assume for example that the screen coordinate system has origin in the upper left-hand corner of the screen, and that the screen is W pixels wide and H pixels high. Then in device-independent coordinates we have $(0,0) \rightarrow (0,H)$ and $(1,1) \rightarrow (W,0)$. So the point (v_1, v_2) in $[0,1] \times [0,1]$ gets mapped to $(v_2 W, 1 - v_1 H)$.

Hidden surface elimination At this point, we assume that we have a collection of triangles (the hull), and a transformation which projects these triangles onto the computer screen. The resulting image would be a "wire-frame" representation of the convex hull. In order to present a more realistic computer image of the hull, we should make invisible those triangles which are on the far side of the hull from the observer. That is, we should eliminate from view the hidden surfaces of the hull.

Triangle sorting One easy approach to this problem is to calculate the coordinates of the center of each triangle, and then calculate the distance from the center of each triangle to the eye point. One then sorts the list of triangles and draws the triangles from-back-to-front, so that the triangle nearest the eye point is drawn last. As each triangle is drawn, it is interior is shaded white with a "flood fill" so that it over-draws the triangles behind it. In this way, hidden surfaces are eliminated.

Back-plane culling Unfortunately, there are some degenerate cases for which this algorithm does not work well. For example, consider the triangles [a,b,c] and [a,b,d] below.



Here, triangle [a,b,c] is in front of triangle [a,b,d], and the algorithm described above would draw these two triangles in the correct order. Now, however, imagine moving the point c downwards towards negative infinity. Then the center of triangle [a,b,c] would also move downwards towards negative infinity, and so the center of triangle [a,b,d] would eventually be closer to our eye. Thus, triangle [a,b,d] would incorrectly be drawn in front of triangle [a,b,c].

Since the object we are drawing is known to be convex, we can make use of a technique called "back plane culling". Again, consider the two triangles above, but also consider the triangles [a,c,d] and [b,c,d] so that we now have a convex figure (a tetrahedron). Note that triangles [a,b,d] and [b,c,d] should not be visible to the eye point, since they are obscured by triangles [a,b,c] and [a,c,d].

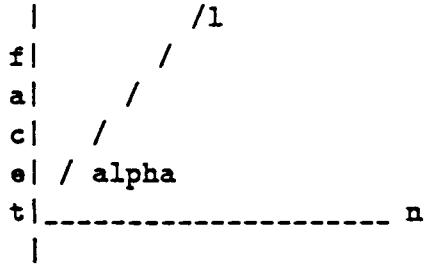
Compute a point e which is in the center of these four triangles; for instance, let the coordinates of e be the average of the coordinates of a,b,c,d. Let $\vec{n}$ be a vector normal to, say, triangle [a,b,c]. Project the eye point onto $\vec{n}$ and project e onto $\vec{n}$. Note that they fall on *opposite* sides of the the vector; that is, the magnitudes of the projections have opposite signs. Then triangle [a,b,c] must be in front of the center point e, and therefore it is a triangle which ***should*** be drawn. Repeating these computations with, say, triangle, [a,b,d], we see that the center point e and the eye point lie on the same side of the triangle, and therefore triangle [a,b,d] should *not* be drawn.

Combining the two This technique of back plane culling only works for single convex figures. In the case of the images rendered by the *hull* program, there is one convex figure as well as three coordinate axes. The coordinate axes themselves are treated as degenerate triangles. After performing back plane culling on the convex figure, the *hull* program also sorts the triangles so that the coordinate axes will be drawn at the right time. That is, they will be partially obscured when they are behind the convex object, and they will be drawn last when they are in front of the convex object. So, *hull* uses a combination of these two hidden surface techniques.

More sophisticated techniques, such as z-buffering, are available, but the above two approaches were chosen for their simplicity and to reduce rendering

time. It should be noted that degenerate cases can still occur. For instance, a coordinate axis might incorrectly appear in front of one facet of the convex hull if the "center" of the axis is closer to the eye than the center of the facet.

Shading At this point we have a fairly realistic image of the convex hull, except that each facet of the hull is shaded the same color (presumably white). Better three dimensional depth clues can be obtained for the user's eye by shading the various facets of the hull. In keeping with the goal of computational simplicity and fast rendering, we adopt the following algorithm. Let point l be the location of an imaginary light source (or lamp) in the object space. For each facet of the convex hull, compute a normal vector $\vec{n}$ as well as a vector $\vec{m}$ whose tail coincides with that of $\vec{n}$ but whose head is at l.



Let α be the angle between these two vectors. If α is near 1, then the lamp is nearly orthogonal to the facet, so the facet is shaded a very light color. If α is near 0, then the lamp is nearly parallel to the facet, so it is painted dark. (Note that if α is near -1, we again paint the facet a light color, so in fact we have two lamps in the object space, diametrically opposite each other.)

Programming with X-Windows Many of the the X-Windows routines used in *hull* are derived from examples found in the book *X Window Applications Programming* by Johnson and Reichard. A more complete treatment can be found in *Introduction to the X Window System* by Oliver Jones. In brief, there are three large sections and one small section of *hull* devoted to dealing with the X-Windows system.

1. The first section deals with actually opening up an X-Window on the DECStation screen. This includes opening the window as well as setting up the colors and the fonts to be used. The files involved here are *windowx.c, colorx.c* and *textx.c*.

2. The second section deals with drawing the convex hull and gadgets into this window. The main file of interest here is *demox.c* which contains the function refreshWindow(). The files *2d.c, 3d.c, hull.c, draw.c,* and *gadgetdata.c* are also involved with drawing the screen, but they do not contain any X-Windows code.

3. The third section deals with waiting for mouse or keyboard events from the user, and taking the appropriate action for each. The main file of interest here is *eventx.c* which contains the function eventLoop().

4. The final, small, section deals with closing the X-Window cleanly. This section is contained in the file *quitx.c.*

Possible improvements to the code Several compromises have been made in the implementation of the *hull* code, either to protect the relative simplicity of the program or to increase the rendering time.

- The hidden surface elimination algorithms could be made more sophisticated, although this would probably increase rendering time.

- The sorting of the triangles includes sorting *all* of the triangles, including those which were culled. Rendering time could be reduced by sorting only the non-culled triangles; this would make the program slightly more complex.

- If the set of point whose hull is to be computed are all co-planar, then the gift-wrapping algorithm tends to create far more triangles than are really necessary to compute the hull. This is considered a degenerate case.

- At the moment, the routine which computes the default location of the eye,see,light-points, called setEye(), needs the information about the convex hull from getData() in order to set the points. But getData() also needs some information from setEye() in order to place the coordinate axes. At the moment, the main() program gets around this by initially calling getData(), then setEye(), and then getData() again. This needs to be fixed, probably by creating a new setAxes() routine.

- Facets of the hull which are nearly edge-on to the observer are not usually rendered in the "hidden surface elimination" mode of the *hull* program. Normally, this does not present a problem. If, however, one uses the axes re-scaling feature, then the absence of nearly edge-on facets could be noticeable, since under the new scaling of the axes these facets are no longer edge-on. Currently, the hidden surface elimination is done before the re-scaling; that is the source of this bug. To fix it, the re-scaling should be done before the hidden surface elimination. Performing the calculations in that order, however, would be more time-consuming for the computer. Since the bug rarely appears, it is not clear that fixing it would be worth the increased computational cost.

```
c Program to compute data for the convex hull generation.

      integer i,nData
      real    minT,maxT,deltaT
      real    t,x,y,z

      nData = 1024
      minT = 0.0
      maxT = 1.0
      deltaT = (maxT-minT)/float(nData-1)

      open(unit=1,file='curve.data')
      write(1,100) nData
  100 format(1x,i4)
      do 10 i = 0,nData-1
        t = float(i)*deltaT + minT
        call eval(t,x,y,z)
        write(1,200) x,y,z
  200 format(1x,3(f12.6,1x))
   10 continue

      end
c-----
      subroutine eval(t,x,y,z)
      real t,x,y,z
      real AA(3,3),xx(3),bb(3)
      real A,B,C,D,b1,b2,b3,kplus,kminus,muplus,muminus

      A  =  30.0
      B  =   0.0
      C  =   0.0
      D  =   2.0
      b1 =   1.0
      b2 =  85.0
      b3 =  50.0
      kminus  =   0.1
      kplus   =   1.5
      muminus =   2.0
      muplus  =  30.0

      AA(1,1) =  D - (kplus - t * (kplus - kminus))
      AA(1,2) = -C
      AA(1,3) =  B

      AA(2,1) =  C
      AA(2,2) = -D - (muplus - t * (muplus - muminus))
      AA(2,3) = -A

      AA(3,1) = -B
      AA(3,2) =  A
      AA(3,3) = -D - (muplus - t * (muplus - muminus))

      bb(1) = -b1
      bb(2) = -b2
      bb(3) = -b3

      call lsarg(3,AA,3,bb,1,xx)

      x = xx(1)
      y = xx(2)
      z = xx(3)

c     test data
c     x = sin(t)+2.0
c     y = sqrt(t)
c     z = t*t/25.0
```

```
c       x = (1.0-t)*sin(16.0*t)
c       y = (1.0-t)*cos(16.0*t)
c       z = (2.0*(t-0.5))**3 + 0.5

        return
        end
```

```
# X-Windows make file
##
## gcc
## cc
COMPILER= cc

EXEC= hull
##
## -O turn on optimizer
## -g turn on debugger
## -Wall nag nag nag
CFLAGS= -g
##
## R4 of the X library:
##LIBS= -lX11 -lm
## R3 of the X library:
LIBS= -L/X11R3 -lX11 -lm
##
OBJECTS=                \
         main.o         \
         2d.o           \
         3d.o           \
         colorx.o       \
         draw.o         \
         eventx.o       \
         gadgetdata.o   \
         getdata.o      \
         hull.o         \
         initx.o        \
         minmax.o       \
         overhead.o     \
         quitx.o        \
         sort.o         \
         textx.o        \
         windowx.o
##
INCLUDES= global.h prototypes.h
##
$(EXEC): $(OBJECTS) $(INCLUDES)
        $(COMPILER) -o $(EXEC) $(OBJECTS) $(LIBS)
##
.c.o: $(INCLUDES)
        $(COMPILER) $(CFLAGS) -c $<
######################
######################
makedata: makedata.f
        f77 -o makedata -u makedata.f -limsl
```

```
/* FILE: prototypes.h
** prototypes */

/* X-specific functions */
void   main(void);
void   refreshWindow(Window theWindow);
void   initX(void);
void   getXInfo(void);
void   setColorWithName(GC theGC,char theName[]);
void   initDefaultColors(void);
void   setColor(GC theGC,int colorNumber);
Window openWindow(int x,int y,int width,int height,int flag,GC *theNewGC);
int    createGC(Window theNewWindow,GC *theNewGC);
XFontStruct *initFont(GC theGC,char fontName[]);
int    eventLoop(void);
void   initEvents(Window theWindow);
void   quitX(void);

/* application functions */
void   getData(void);
void   chull(void);
void   writeData(void);
void   setEye(void);
float  max4(float w,float x,float y,float z);
float  min4(float w,float x,float y,float z);
float  max3(float w,float x,float y);
float  min3(float w,float x,float y);
float  max2(float w,float x);
float  min2(float w,float x);
int    altTop(float x);
int    viewLeft(float x);
int    viewTop(float x);
float  fromAltTop(int i);
float  fromViewLeft(int i);
float  fromViewTop(int i);
void   gadgetData(void);
void   sortTriangles(int doDraw[],int sorted[]);
int    whichDraw(int doDraw[]);
int    getShade(int i);
int    isAnEdge(int p1,int p2,intTriangle T);

/* sorting functions */
int    partition(float values[],int index[],int i,int j,int pindex);
void   quickSort(float values[],int index[],int first,int last);
void   bubbleSort(float values[],int index[],int first,int last);
int    findPivot(float values[],int index[],int i,int j);
void   swap(float values[],int index[],int l,int r);

/* 2D graphics functions */
dvdCoord dvi2dvd(dviCoord theDviCoord);
dviCoord vr2dvi(vrCoord theVrCoord);
void   dvdDrawLine(Window theWindow,GC theGC,dvdCoord dvdp,dvdCoord dvdq);
void   dvdDrawRectangle(Window theWindow,GC theGC,dvdCoord dvdp,dvdCoord dvdq);
void   dvdFillRectangle(Window theWindow,GC theGC,dvdCoord dvdp,dvdCoord dvdq);
void   dvdDrawGadget(Window theWindow,GC theGC,int whichGadget);
void   dviDrawLine(Window theWindow,GC theGC,dviCoord dvip,dviCoord dviq);
void   vrDrawLine(Window theWindow,GC theGC,vrCoord vrp,vrCoord vrq);
void   dvdDrawPoint(Window theWindow,GC theGC,dvdCoord dvdp);
void   vrDrawPoint(Window theWindow,GC theGC,vrCoord vrp);
void   dvdPrint(Window theWindow,GC theGC,dvdCoord dvdp,char theString[]);
void   vrPrint(Window theWindow,GC theGC,vrCoord vrp,char theString[]);

/* 3D graphics functions */
vrCoord os2vr(osCoord theOsCoord);
void   osDrawLine(Window theWindow,GC theGC,osCoord osp,osCoord osq);
void   initShading(void);
```

```
void   osDrawTriangle(Window theWindow,GC theGC,triangle theTriangle,
       int theColor);
void   osDrawPoint(Window theWindow,GC theGC,osCoord osp);
void   osPrint(Window theWindow,GC theGC,osCoord osp,char theString[]);
void   PSInit(void);
void   initPP(void);
void   initT(void);
void   initR(void);
void   cross(float u[],float v[],float w[]);
void   normalize(float u[],float v[]);
float  norm(float v[]);
float  dotprd(float u[],float v[]);
void   PSPlot(float o[],float vr[]);
void   permute(float o[]);
void   unpermute(float o[]);
void   doPP(float o[],float ov[]);
void   doT(float ov[],float v[]);
void   doR(float v[],float vr[]);

void   rect2sphere(float rect[],float sphere[]);
void   sphere2rect(float sphere[],float rect[]);

/* convex hull functions */
int      paramCurve(osCoord theData[],float minT,float maxT,int n);
int      hull3d(osCoord theData[],triangle theHull[],intTriangle intHull[],int n
int      GiftWrapping(osCoord theData[],triangle theHull[],intTriangle intHull[]
Triangle findFirstFacet(osCoord theData[],triangle theHull[],int freq[],
         int numD);
int      addToFile(Triangle F,Edge T[],int numT);
int      push(Triangle F,Triangle Q[],int numQ);
Triangle pop(Triangle Q[],int numQ);
void     getEdges(Triangle F,Edge t[]);
int      isCommon(Edge at,Edge T[],int numT);
Triangle giftWrap(Edge e,Triangle F,osCoord theData[],int freq[],int numD);
int      insertDelete(Triangle F,Edge T[],int numT);
int      storeh(Triangle F,triangle theHull[],intTriangle intHull[],osCoord theD
```

```
/* FILE: global.h
** Structures pertaining to this specific application */

typedef struct {  /* device dependent coordinate system */
  int horizontal;
  int vertical;
  } dvdCoord;

typedef struct {  /* device independent coordinate system */
  float x;
  float y;
  } dviCoord;

typedef struct {  /* view-rectangle coordinate system */
  float x;
  float y;
  } vrCoord;

typedef struct { /* object space coordinates */
  float x;
  float y;
  float z;
  } osCoord;

typedef struct { /* a triangle in object space */
  float v1x; float v1y; float v1z;
  float v2x; float v2y; float v2z;
  float v3x; float v3y; float v3z;
  float centerx; float centery; float centerz;
  float normalx; float normaly; float normalz;
  } triangle;

typedef struct { /* the data numbers of a triangle on the curve */
  int p1;
  int p2;
  int p3;
  } intTriangle;

typedef struct { int vertex[4]; } Triangle;
typedef struct { int vertex[3]; } Edge;

typedef struct {
  int windowWidth;      /* window parameters */
  int windowHeight;
  int rightBarWidth;    /* menu bar parameters */
  int topBarHeight;
  int dvdWidth;         /* device dependent coordinate parameters */
  int dvdHeight;
  float vrLeft;         /* view-rectangle coordinate parameters */
  float vrRight;
  float vrTop;
  float vrBottom;
  float osLeft;         /* object-space coordinate parameters */
  float osRight;
  float osTop;
  float osBottom;
  float osFront;
  float osBack;
  float vcLeft;         /* object-space-view-cube coordinate parameters */
  float vcRight;
  float vcTop;
  float vcBottom;
  float vcFront;
  float vcBack;
  float rho;            /* various transformation parameters */
  float tilt;
```

```
  float c[4];
  float A[3][4];
  float beta;
  float eye[4];
  float see[4];
  float light[4];
  float Rotate[3][3];
  int   plotType;
  int   solidType;
  int   shadingColor[66];
  float x_scale;
  float y_scale;
  float z_scale;
  } coordParams;

typedef struct {
  int  top;
  int  bottom;
  int  left;
  int  right;
  char string[80];
  } gadget;

/* gadget numbers */
#define QUIT     0
#define CCWISE   1
#define CLWISE   2
#define OVMAG    3
#define OVMIN    4
#define OVHEAD   5
#define ALT      6
#define ALTMAG   7
#define ALTMIN   8
#define INWARD   9
#define OUTWARD 10
#define VIEWIN  11
#define VIEWOUT 12
#define OVER    13
#define UNDER   14
#define AXESG   15
#define WIREG   16
#define HIDEG   17
#define SHADG   18
#define COARSE  19
#define FINE    20
#define MINDM   21
#define MINDP   22
#define MAXDM   23
#define MAXDP   24
#define HIGHLT  25
#define WRITEDT 26
#define X2      27
#define X5      28
#define Y2      29
#define Y5      30
#define Z2      31
#define Z5      32
#define DOHULL  33
#define NUMGADGETS 33

#define STRLEN    80
#define MAXDATA      1024
#define MAXTRIANGLES 4096
#define MAXEDGES     2048

/* plotting axes types */
```

```
#define MATHPLOT  1
#define CSPLOT    2

/* how to draw the surface */
#define AXES      0
#define WIREFRAME 1
#define HIDDEN    2
#define SHADED    3

#define NUMSHADES 66

/* gadget size parameters */
#define gadgetTall   16
#define gadgetBorder  8
#define gadgetWide   57
```

```
/* FILE: textx.c
** Text rendering routines. */

/* X-windows include files: */
#include <X11/Xlib.h>

#include "global.h"
#include "prototypes.h"

/* Global variables: */
extern Display *theDisplay;


  /* Initialize a font. */
  XFontStruct *initFont(GC theGC,char fontName[])
  {
    XFontStruct *fontStruct;
    fontStruct = XLoadQueryFont(theDisplay,fontName);
    if (fontStruct != 0) { XSetFont(theDisplay,theGC,fontStruct->fid); }
    return(fontStruct);
  }/* end function initFont() */
```

```
/* FILE: windowx.c
** Put up a window. */

/* X-windows include files: */
#include <X11/Xlib.h>
#include <X11/Xutil.h>

/* Standard I/O include file: */
#include <stdio.h>

#include "global.h"
#include "prototypes.h"

/* The bitmap file for the application's icon: */
#include "theIcon"

/* Global variables: */
extern Display *theDisplay;
extern int      theScreen;
extern int      theDepth;
extern unsigned long theBlackPixel;
extern unsigned long theWhitePixel;
extern coordParams theCoordParams;

#define BORDER_WIDTH 2
#define WINDOW_TITLE "Convex Hull"



  /* Function to open a window. */
  Window openWindow(int x,int y,int width,int height,int flag,GC *theNewGC)

    XSetWindowAttributes theWindowAttributes;
    XSizeHints           theSizeHints;
    XClassHint           theClassHint;
    unsigned long        theWindowMask;
    Window               theNewWindow;
    Pixmap               theIconPixmap;
    XWMHints             theWMHints;

    /* Figure out how big the window should be.  If the user asked for width
    ** or depth = -1, they want the window to be as big as possible. */
    if (width  == -1) { width  = DisplayWidth(theDisplay,theScreen);  }
    if (height == -1) { height = DisplayHeight(theDisplay,theScreen); }

    /* for our application, this information needs to be made global too: */
    theCoordParams.windowWidth  = width;
    theCoordParams.windowHeight = height;

    /* Define the window's attributes. */
    theWindowAttributes.border_pixel      = BlackPixel(theDisplay,theScreen);
    theWindowAttributes.background_pixel = WhitePixel(theDisplay,theScreen);
    theWindowAttributes.override_redirect = False;
    theWindowMask = CWBackPixel | CWBorderPixel | CWOverrideRedirect;

    /* Create a window definition on the display. */
    theNewWindow = XCreateWindow(theDisplay,RootWindow(theDisplay,theScreen),
      x,y,width,height,BORDER_WIDTH,theDepth,InputOutput, CopyFromParent,
      theWindowMask,&theWindowAttributes);

    /* Convert the icon file into Pixmap format. */
    theIconPixmap = XCreateBitmapFromData(theDisplay,theNewWindow,theIcon_bits,
      theIcon_width,theIcon_height);

    /* Define the icon to be associated with this application (window). */
```

```
    theWMHints.icon_pixmap    = theIconPixmap;
    theWMHints.initial_state = NormalState;
    theWMHints.flags          = IconPixmapHint | StateHint;
    XSetWMHints(theDisplay,theNewWindow,&theWMHints);

    /* Define the application icon name. */
    XSetIconName(theDisplay,theNewWindow,WINDOW_TITLE);

    /* Define the class and name of the application (window). */
    theClassHint.res_name  = WINDOW_TITLE;
    theClassHint.res_class = WINDOW_TITLE;
    XSetClassHint(theDisplay,theNewWindow,&theClassHint);

    /* Define the window's desired size and position. */
    theSizeHints.flags  = PPosition | PSize;
    theSizeHints.x      = x;
    theSizeHints.y      = y;
    theSizeHints.width  = width;
    theSizeHints.height = height;
    XSetNormalHints(theDisplay,theNewWindow,&theSizeHints);

    /* Create a graphics context (GC) for the window. See below. */
    if (createGC(theNewWindow,theNewGC) == 0) {
      XDestroyWindow(theDisplay,theNewWindow);
      return((Window) 0);
    }/* end if */

    /* Now that the window is defined, map it to the screen. */
    XMapWindow(theDisplay,theNewWindow);

    /* Flush out all of the queued up X-requests to the X-server. */
    XFlush(theDisplay);

    return(theNewWindow);

  }/* end function openWindow() */



  /* Create a graphics context (GC) for the window. */
  int createGC(Window theNewWindow,GC *theNewGC)
  {
    XGCValues theGCValues;

    *theNewGC = XCreateGC(theDisplay,theNewWindow,(unsigned long) 0,
      &theGCValues);

    if (*theNewGC == 0) { return(0); }   /* error: unable to create a GC */
    else {
      XSetForeground(theDisplay,*theNewGC,theBlackPixel);
      XSetBackground(theDisplay,*theNewGC,theWhitePixel);
      return(1);
    }/* end if */

  }/* end function createGC() */
```

```
/* FILE: sort.c
** Routines for sorting. */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "global.h"
#include "prototypes.h"


  /* function bubbleSort */
  void bubbleSort(float values[],int index[],int first,int last)
  {
    int i,j;
    for (i = first; i <= last-1; ++i) {
      for (j = last; j >= i+1; --j) {
        if (values[j-1] > values[j]) {
          swap(values,index,j-1,j);
        }/* end if */
      }/* end for */
    }/* end for */
  }/* end function bubbleSort() */



  /* function quickSort */
  void quickSort(float values[],int index[],int i,int j)
  {
    int   pindex,k;
    if (j <= i) { return; }
    if ((j-i) < 9) {
      bubbleSort(values,index,i,j);
    } else {
      pindex = findPivot(values,index,i,j);
      if (pindex != 0) {
        k = partition(values,index,i,j,pindex);
        quickSort(values,index,i,k-1);
        quickSort(values,index,k,j);
      }/* end if */
    }/* end if */
  }/* end function quickSort() */


  int partition(float values[],int index[],int i,int j,int pindex)
  {
    float pivot;
    int   l,r;
    pivot = values[pindex];
    l = i; r = j;
    do {
      swap(values,index,l,r);
      while (values[l]  < pivot) { l++; }
      while (values[r] >= pivot) { r--; }
    } while (l <= r);
    return(l);
  }/* end function partition() */


  int findPivot(float values[],int index[],int i,int j)
  {
    float firstkey;
    int   k;
```

```
  firstkey = values[i];
  for (k = i+1; k <= j; ++k) {
    if (values[k] > firstkey) { return(k); }
    else { if (values[k] < firstkey) { return(i); } }
  }/* end for */
  return(0);
}/* end function */


void swap(float values[],int index[],int l,int r)
{
  int   t;
  float v;
  t = index[l];  index[l] = index[r];   index[r] = t;
  v = values[l]; values[l] = values[r]; values[r] = v;
}/* end function swap() */
```

```
/* FILE: 2d.c
** Routines to perform 2D (and some 3D) coordinate conversions. */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <math.h>
#include "global.h"
#include "prototypes.h"

extern coordParams  theCoordParams;


  /* This function converts a device independent (DVI) coordinate into
  ** a device dependent (DVD) one.  Note that the DVD display area is the
  ** lower left-hand corner of our window, as the top and right side of the
  ** window are reserved for drawing gadgets into */
  dvdCoord dvi2dvd(dviCoord theDviCoord)
  {
    dvdCoord theDvdCoord;
    theDvdCoord.horizontal =
      (int)( theDviCoord.x       *(float)theCoordParams.dvdWidth );
    theDvdCoord.vertical   = theCoordParams.topBarHeight +
      (int)((1.0 - theDviCoord.y)*(float)theCoordParams.dvdHeight);
    return(theDvdCoord);
  }/* end function dvi2dvd() */



  /* This function converts a view-rectangle (VR) coordinate into a DVI
  ** coordinate. */
  dviCoord vr2dvi(vrCoord theVrCoord)
  {
    dviCoord theDviCoord;
    theDviCoord.x = (theVrCoord.x-theCoordParams.vrLeft) /
      (theCoordParams.vrRight-theCoordParams.vrLeft);
    theDviCoord.y = (theVrCoord.y-theCoordParams.vrBottom) /
      (theCoordParams.vrTop-theCoordParams.vrBottom);
    return(theDviCoord);
  }/* end function vr2dvi */



  /* This function converts an object-space (OS) coordinate into a view-
  ** rectangle coordinate. Note that either MathInit or CSInit must be
  ** called first to initialize the various transformations.  */
  vrCoord os2vr(osCoord theOsCoord)
  {
    vrCoord theVrCoord;
    float o[4],vr[3];
    o[1] = theOsCoord.x;
    o[2] = theOsCoord.y;
    o[3] = theOsCoord.z;
    PSPlot(o,vr);
    theVrCoord.x = vr[1];
    theVrCoord.y = vr[2];
    return(theVrCoord);
  }/* end function vr2dvi */



  /* Convert rectangular coordinates to spherical. */
```

```
void rect2sphere(float rect[],float sphere[])
{
   float theta; /* around latitude lines */
   float phi;   /* down longitude lines */
   float rho;   /* radius */
   float x,y,z;

   x = rect[1]; y = rect[2]; z = rect[3];
   theta = acos(x/sqrt(x*x+y*y));
   if (y < 0.0) { theta = -theta; }
   rho = sqrt(x*x+y*y+z*z);
   phi = acos(z/rho);
   sphere[1] = theta; sphere[2] = phi; sphere[3] = rho;

}/* end function */



/* Convert rectangular coordinates to spherical. */
void sphere2rect(float sphere[],float rect[])
{
   float theta; /* around latitude lines */
   float phi;   /* down longitude lines */
   float rho;   /* radius */
   float x,y,z;
   static float pi = 3.1415926;

   theta = sphere[1]; phi = sphere[2]; rho = sphere[3];
   if (phi < 0.0) { phi = 0.01; }
   if (phi > pi ) { phi = pi-0.01;  }
   x = rho*cos(theta)*sin(phi);
   y = rho*sin(theta)*sin(phi);
   z = rho*cos(phi);
   rect[1] = x; rect[2] = y; rect[3] = z;

}/* end function */
```

```
/* FILE: 3d.c
** Routines to perform basic 3d -> 2d coordinate conversion. */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <math.h>
#include "global.h"
#include "prototypes.h"

extern coordParams  theCoordParams;


  /* initialize everything for graphics-standard (z=inward) plotting */
  void PSInit()
  {
    float temp1[4],temp2[4];
    int   i;
    permute(theCoordParams.eye);   /* permute the data if necessary, i.e. if */
    permute(theCoordParams.see);   /* we're in MATHPLOT mode, where z=upward */
    permute(theCoordParams.light);
    initPP();          /* initialize the data for projecting onto view-plane */
    initT();           /* initialize data for converting to view-plane coords */
    initR();           /* initialize data for rotation due to head tilt */
    unpermute(theCoordParams.eye);  /* un-do the permutations from above */
    unpermute(theCoordParams.see);
    unpermute(theCoordParams.light);
    unpermute(theCoordParams.c);
    /* also un-permute the conversion matrix A, used by the T transformation */
    for (i=1; i<= 3; ++i) {
      temp1[i] = theCoordParams.A[1][i];
      temp2[i] = theCoordParams.A[2][i];
    }/* end for */
    unpermute(temp1); unpermute(temp2);
    for (i=1; i<=3; ++i) {
      theCoordParams.A[1][i]=temp1[i];
      theCoordParams.A[2][i]=temp2[i];
    }/* end for */
  }/* end function */



  /* initialize the perspective projection */
  void initPP()
  {
    float t[4];
    int   i;

    /* c = rho * see + (1 - rho) * eye */
    for (i = 1; i <= 3; ++i) {
      theCoordParams.c[i] = theCoordParams.rho * theCoordParams.see[i] +
        (1.0 - theCoordParams.rho) * theCoordParams.eye[i];
    }/* end for */

    /* beta = dot(c-eye,c-eye)
    ** By storing the value beta, we save some computations later. */
    for (i = 1; i <= 3; ++i) {
      t[i] = theCoordParams.c[i]-theCoordParams.eye[i];
    }/* end for */
    theCoordParams.beta = dotprd(t,t);

  }/* end function */
```

```
/* initialize the array A that is used by doT; A converts
** from 3D object space to 2D view plane coordinates */
void initT()

  float upward[4],d1[4],d2[4],v1[4],v2[4],t1[4],t2[4];
  int   i;

  /* find the up-direction emanating from see */
  for (i = 1; i <= 3; ++i) {
    upward[i] = theCoordParams.see[i];
  }/* end for */
  upward[2]+=1.0;

  /* let t1 = c-see, normalized; t1 is perpendicular to the view plane */
  for (i = 1; i <= 3; ++i) {
    t1[i] = theCoordParams.c[i]-theCoordParams.see[i];
  }/* end for */
  normalize(t1,t1);

  /* project the see-plus-unit-y-vector onto the view plane in order
  ** to start finding the view plane y-axis */
  doPP(upward,d2);

  /* let v2 = d2-c,normalized; t2 is parallel to the view plane y-axis */
  for (i = 1; i <= 3; ++i) {
    t2[i] = d2[i]-theCoordParams.c[i];
  }/* end for */
  normalize(t2,v2);

  /* find a vector that is perpendicular to both t1 and t2; this must
  ** be the view plane x-axis */
  cross(t1,t2,d1);
  normalize(d1,v1);

  /* Form A; the first row of A is v1 and the second row is v2.
  ** note then that A*v1=<1,0> and A*v2=<0,1>.  Then to convert from
  ** 3D to 2D coordinates will only require A*(v-c). */
  for (i = 1; i <= 3; ++i) {
    theCoordParams.A[1][i] = v1[i];
    theCoordParams.A[2][i] = v2[i];
  }/* end for */

}/* end function */



/* initialize the rotation matrix Rotate */
void initR()
{
  theCoordParams.Rotate[1][1] =  cos(theCoordParams.tilt);
  theCoordParams.Rotate[1][2] =  sin(theCoordParams.tilt);
  theCoordParams.Rotate[2][1] = -sin(theCoordParams.tilt);
  theCoordParams.Rotate[2][2] =  cos(theCoordParams.tilt);
}/* end function */



/* compute the (left handed) cross product of vectors u and v */
void cross(float u[],float v[],float w[])
{
  w[1] = -u[2]*v[3]+v[2]*u[3];
  w[2] =  u[1]*v[3]-v[1]*u[3];
```

```
  w[3] = -u[1]*v[2]+v[1]*u[2];
}/* end function */


/* normalize a vector u to get v */
void normalize(float u[],float v[])
{
  float normu;
  int   i;
  normu = norm(u);
  if (normu != 0.0) {
    for (i = 1; i <= 3; ++i) {
      v[i] = u[i]/normu;
    }/* end for */
  }/* end if */
}/* end function */



/* compute the Euclidean norm of v */
float norm(float v[])
{
  float s;
  s = sqrt(dotprd(v,v));
  return(s);
}/* end function */


/* compute the dot product of u and v */
float dotprd(float u[],float v[])
{
  float d;
  int   i;
  d = 0.0;
  for (i = 1; i <= 3; ++i) {
    d+= u[i] * v[i];
  }/* end for */
  return(d);
}/* end function */



/* The routines above are all system initializations.
**
** The routines below actually perform the transformations. */



/* convert from 3d object-space coordinate system to view-plane coords */
void PSPlot(float o[],float vr[])
{
  float ov[4],v[3];

  /* re-scale the points */
  o[1]*=theCoordParams.x_scale;
  o[2]*=theCoordParams.y_scale;
  o[3]*=theCoordParams.z_scale;
  /* project onto the view plane in 3D coordinates */
  doPP(o,ov);
```

```
  /* convert to view plane natural coordinates centered on point c */
  doT(ov,v);
  /* perform the rotation due to theCoordParams.tilt */
  doR(v,vr);

* end function*/



/* permute a vector from (x,y,z) to (z,x,y) */
void permute(float o[])
{
  float  t[4];
  if (theCoordParams.plotType == MATHPLOT) {
    t[1] = o[2];
    t[2] = o[3];
    t[3] = o[1];
    o[1] = t[1];
    o[2] = t[2];
    o[3] = t[3];
  }/* end if */
}/* end function */



/* unpermute a vector from (x,y,z) to (z,x,y) */
void unpermute(float o[])
{
  float  t[4];
  if (theCoordParams.plotType == MATHPLOT) {
    t[2] = o[1];
    t[3] = o[2];
    t[1] = o[3];
    o[1] = t[1];
    o[2] = t[2];
    o[3] = t[3];
  }/* end if */
}/* end function */



/* project from 3d object-space onto the view plane in object-space
** coordinate system */
void doPP(float o[],float ov[])
{
  float alpha,t1[4],t2[4];
  int   i;

  /* alpha = dot(c-eye,c-eye)/dot(o-eye,c-eye)
  ** v = alpha * o + (1 - alpha) *eye */

  for (i = 1; i <= 3; ++i) {
    t1[i] = theCoordParams.c[i]-theCoordParams.eye[i];
    t2[i] = o[i]-theCoordParams.eye[i];
  }/* end for */
  alpha = theCoordParams.beta/dotprd(t2,t1);
  for (i = 1; i <= 3; ++i) {
    ov[i] = alpha*o[i]+(1.0-alpha)*theCoordParams.eye[i];
  }/* end for */

}/* end function */
```

```
/* convert a point on the view plane from its object-space
   coordinate system into the view plane's coordinate system */
void doT(float ov[],float v[])
{
  int i,j;

  /* T(ov) = A * (ov - c) */
  for (i = 1; i <= 2; ++i) {
    v[i] = 0.0;
  }/* end for */
  for (j = 1 ; j <= 3; ++j) {
    for (i = 1; i <= 2; ++i) {
      v[i]+= theCoordParams.A[i][j] * (ov[j]-theCoordParams.c[j]);
    }/* end for */
  }/* end for */

}/* end function */



/* perform the rotation due to head tilt */
void doR(float v[],float vr[])
{
  vr[1] = theCoordParams.Rotate[1][1]*v[1]+theCoordParams.Rotate[1][2]*v[2];
  vr[2] = theCoordParams.Rotate[2][1]*v[1]+theCoordParams.Rotate[2][2]*v[2];
}/* end function */
```

```
/* FILE: colorx.c
** Set up the application colors; be able to specify colors by number or by
** name, for the first 66 X-Windows named colors */

/* X-windows include files: */
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <stdio.h>

#include "global.h"
#include "prototypes.h"
extern coordParams theCoordParams;

/* Global variables: */
extern Display *theDisplay;
extern int      theDepth;
extern unsigned long theBlackPixel;
extern unsigned long theWhitePixel;
extern Colormap theColormap;

/* Set up English text for colors. */
#define maxPixels 132
#define stdColors 66
unsigned long  thePixels[maxPixels];

  char          *theColorNames[stdColors] =
    {"Aquamarine",          /* 00 */
     "Black",               /* 01 */
     "Blue",                /* 02 */
     "BlueViolet",          /* 03 */
     "Brown",               /* 04 */
     "CadetBlue",           /* 05 */
     "Coral",               /* 06 */
     "CornflowerBlue",      /* 07 */
     "Cyan",                /* 08 */
     "DarkGreen",           /* 09 */
     "DarkOliveGreen",      /* 10 */
     "DarkOrchid",          /* 11 */
     "DarkSlateBlue",       /* 12 */
     "DarkSlateGrey",       /* 13 */
     "DarkTurquoise",       /* 14 */
     "DimGrey",             /* 15 */
     "Firebrick",           /* 16 */
     "ForestGreen",         /* 17 */
     "Gold",                /* 18 */
     "Goldenrod",           /* 19 */
     "Grey",                /* 20 */
     "Green",               /* 21 */
     "GreenYello",          /* 22 */
     "IndianRed",           /* 23 */
     "Khaki",               /* 24 */
     "LightBlue",           /* 25 */
     "LightGrey",           /* 26 */
     "LightSteelBlue",      /* 27 */
     "LimeGreen",           /* 28 */
     "Magenta",             /* 29 */
     "Maroon",              /* 30 */
     "MediumAquamarine",    /* 31 */
     "MediumBlue",          /* 32 */
     "MediumForestGreen",   /* 33 */
     "MediumGoldenrod",     /* 34 */
     "MediumOrchid",        /* 35 */
     "MediumSeaGreen",      /* 36 */
     "MediumSlateBlue",     /* 37 */
     "MediumSpringGreen",   /* 38 */
     "MediumTurquoise",     /* 39 */
```

```
    "MediumVioletRed",      /* 40 */
    "MidnightBlue",         /* 41 */
    "Navy",                 /* 42 */
    "Orange",               /* 43 */
    "OrangeRed",            /* 44 */
    "Orchid",               /* 45 */
    "PaleGreen",            /* 46 */
    "Pink",                 /* 47 */
    "Plum",                 /* 48 */
    "Red",                  /* 49 */
    "Salmon",               /* 50 */
    "SeaGreen",             /* 51 */
    "Sienna",               /* 52 */
    "SkyBlue",              /* 53 */
    "SlateBLue",            /* 54 */
    "SpringGreen",          /* 55 */
    "SteelBlue",            /* 56 */
    "Tan",                  /* 57 */
    "Thistle",              /* 58 */
    "Turquoise",            /* 59 */
    "Violet",               /* 60 */
    "VioletRed",            /* 61 */
    "Wheat",                /* 62 */
    "White",                /* 63 */
    "Yellow",               /* 64 */
    "YellowGreen"};         /* 65 */


/* This function sets the GC with the foreground color named. */
void setColorWithName(GC theGC,char theName[])

{
  int i;
  i = 0;
  while((strcmp(theName,theColorNames[i])!=0)&&(i<stdColors)) { i++; }
  if (i < stdColors) { XSetForeground(theDisplay,theGC,thePixels[i]); }
}/* end function setColorWithName() */



/* Attempt to set up a local color table with the default X11 colors. */
void initDefaultColors()
{
  XColor theRGBColor,theHardwareColor;
  int    theStatus;
  char   theString[80];
  unsigned int i,h,r,g,b,top,bottom;

  if (theDepth > 1) {
    /* use the 66 standard colors */
    for (i=0; i<stdColors; i++) {
      theStatus = XLookupColor(theDisplay,theColormap,theColorNames[i],
        &theRGBColor,&theHardwareColor);
      if (theStatus != 0) {
        theStatus = XAllocColor(theDisplay,theColormap,&theHardwareColor);
        if (theStatus != 0) {
          thePixels[i] = theHardwareColor.pixel;
        } else {
          thePixels[i] = theBlackPixel;
        }/* end if */
      }/* end if */
    }/* end for */
    /* and also create a 66 color grey scale */
    for (i=stdColors; i<=maxPixels-1; i++) {
```

```
      h = i-stdColors;
      bottom = 16; top = 255-16;          /* range of rgb color values */
      h = bottom + ((top)-bottom)*h/66;   /* h ranges from top to bottom */
      r = h;
      g = h;
      b = h;
      sprintf(theString,"#%2x%2x%2x"   ,r,g,b);
      theStatus = XParseColor(theDisplay,theColormap,theString,
        &theRGBColor);
      if (theStatus != 0) {
        theStatus = XAllocColor(theDisplay,theColormap,&theRGBColor);
        if (theStatus != 0) {
          thePixels[i] = theRGBColor.pixel;
        } else {
          thePixels[i] = theBlackPixel;
        }/* end if */
      }/* end for */
    } else {
      for (i=0; i<stdColors; i++) {
        if (strcmp("White",theColorNames[i]) == 0) {
          thePixels[i] = theWhitePixel;
        } else {
          thePixels[i] = theBlackPixel;
        }/* end if */
      }/* end for */
    }/*end if */
  }/* end function initDefaultColors() */



  /* Set the graphic context (GC) to have a foreground color of colorNumber */
  void setColor(GC theGC,int colorNumber)
  {
    if ((colorNumber < maxPixels) && (colorNumber >= 0 ))
      { XSetForeground(theDisplay,theGC,thePixels[colorNumber]); }
  }/* end function setColor() */
```

```
/* FILE: quitx.c
** Close down X-Windows. */

/* X-windows include files: */
#include <X11/Xlib.h>
#include <X11/Xutil.h>

#include "global.h"
#include "prototypes.h"

extern Display *theDisplay;


  void quitX()
  {
    XCloseDisplay(theDisplay);
  }/* end of function quitX() */
```

```
/* FILE: minmax.c
** Routines for computing mins and maxs. */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <math.h>
#include "global.h"
#include "prototypes.h"



/* function max of four elements */
float max4(float w,float x,float y,float z)
{
  float m;
  m = w;
  if (x > m) { m = x; }
  if (y > m) { m = y; }
  if (z > m) { m = z; }
  return(m);
}/* end function max() */



/* function min of four elements*/
float min4(float w,float x,float y,float z)
{
  float m;
  m = w;
  if (x < m) { m = x; }
  if (y < m) { m = y; }
  if (z < m) { m = z; }
  return(m);
}/* end function min() */



/* function max of three elements */
float max3(float w,float x,float y)
{
  float m;
  m = w;
  if (x > m) { m = x; }
  if (y > m) { m = y; }
  return(m);
}/* end function max() */



/* function min of three elements*/
float min3(float w,float x,float y)
{
  float m;
  m = w;
  if (x < m) { m = x; }
  if (y < m) { m = y; }
  return(m);
}/* end function min() */
```

```
/* function max of two elements */
float max2(float w,float x)
{
  float m;
  m = w;
  if (x > m) { m = x; }
  return(m);
}/* end function max() */



/* function min of two elements*/
float min2(float w,float x)
{
  float m;
  m = w;
  if (x < m) { m = x; }
  return(m);
}/* end function min() */
```

```
/* FILE: overhead.c
** Routines to perform computations for the overhead-view gadgets. */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <math.h>
#include "global.h"
#include "prototypes.h"

extern coordParams theCoordParams;
extern gadget      theGadgets[80];
extern int         drawGadgets;


  /* function to compute altimeter coordinates from object space coordinate */
  int altTop(float x)
  {
    int i;
    i = theGadgets[ALT].top +
        (int)( (theCoordParams.osTop-x)
        *(float)(theGadgets[ALT].bottom - theGadgets[ALT].top)
        /(theCoordParams.osTop-theCoordParams.osBottom) );
    return(i);
  }/* end function altTop() */



  /* function to compute view-screen left coordinates */
  int viewLeft(float x)
  {
    int i;
    i = theGadgets[OVHEAD].left +
        (int)( (x-theCoordParams.osLeft)
        *(float)(theGadgets[OVHEAD].right - theGadgets[OVHEAD].left)
        /(theCoordParams.osRight-theCoordParams.osLeft) );
    return(i);
  }/* end function viewLeft() */



  /* function to compute view-screen top coordinates */
  int viewTop(float x)
  {
    int i;
    i = theGadgets[OVHEAD].bottom -
        (int)( (theCoordParams.osTop-x)
        *(float)(theGadgets[OVHEAD].bottom - theGadgets[OVHEAD].top)
        /(theCoordParams.osTop-theCoordParams.osBottom) );
    return(i);
  }/* end function viewTop() */



  /* function to compute from altimeter coordinates */
  float fromAltTop(int i)
  {
    float x;
    x = theCoordParams.osTop -
      (float)(i-theGadgets[ALT].top)
      *(theCoordParams.osTop-theCoordParams.osBottom)
      /(float)(theGadgets[ALT].bottom-theGadgets[ALT].top);
    return(x);
  }/* end function fromAltTop() */
```

```
/* function to compute from view-screen left coordinates */
float fromViewLeft(int i)
{
  float x;
  x = theCoordParams.osLeft -
    (float)(i-theGadgets[OVHEAD].left)
    *(theCoordParams.osRight-theCoordParams.osLeft)
    /(float)(theGadgets[OVHEAD].left-theGadgets[OVHEAD].right);
  return(x);
}/* end function fromViewLeft() */



/* function to compute from view-screen top coordinates */
float fromViewTop(int i)
{
  float x;
  x = theCoordParams.osBack +
    (float)(i-theGadgets[OVHEAD].top)
    *(theCoordParams.osFront-theCoordParams.osBack)
    /(float)(theGadgets[OVHEAD].bottom-theGadgets[OVHEAD].top);
  return(x);
}/* end function fromViewTop() */
```

```
/* FILE: draw.c
** Routines to perform drawing functions. */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "global.h"
#include "prototypes.h"

extern Display     *theDisplay;
extern coordParams  theCoordParams;
extern gadget       theGadgets[80];



  /* Draw a line from p to q which has been presented in dvd coordinates. */
  void dvdDrawLine(Window theWindow,GC theGC,dvdCoord dvdp,dvdCoord dvdq)
  {
    XDrawLine(theDisplay,theWindow,theGC,dvdp.horizontal,dvdp.vertical,
      dvdq.horizontal,dvdq.vertical);
  }/* end function dvdDrawLine() */



  /* Draw a rectangle, p to q ,which has been presented in dvd coordinates. */
  void dvdDrawRectangle(Window theWindow,GC theGC,dvdCoord dvdp,dvdCoord dvdq)
  {
    int height,width;
    height = abs(dvdq.vertical  -dvdp.vertical);
    width  = abs(dvdq.horizontal-dvdp.horizontal);
    XDrawRectangle(theDisplay,theWindow,theGC,dvdp.horizontal,dvdp.vertical,
      width,height);
  }/* end function dvdDrawRectangle() */



  /* Draw a fill rect, p to q ,which has been presented in dvd coordinates. */
  void dvdFillRectangle(Window theWindow,GC theGC,dvdCoord dvdp,dvdCoord dvdq)
  {
    int height,width;
    height = dvdq.vertical  -dvdp.vertical;
    width  = dvdq.horizontal-dvdp.horizontal;
    XFillRectangle(theDisplay,theWindow,theGC,dvdp.horizontal,dvdp.vertical,
      width,height);
  }/* end function dvdFillRectangle() */



  /* Draw a gadget, p to q ,which has been presented in dvd coordinates. */
  void dvdDrawGadget(Window theWindow,GC theGC,int whichGadget)
  {
    int       height,width;
    dvdCoord dvdp,dvdq;
    char      theString[STRLEN];

    dvdp.vertical   = theGadgets[whichGadget].top;
    dvdp.horizontal = theGadgets[whichGadget].left;
    dvdq.vertical   = theGadgets[whichGadget].bottom;
    dvdq.horizontal = theGadgets[whichGadget].right;
```

```
  strcpy(theString,theGadgets[whichGadget].string);

  height = dvdq.vertical   - dvdp.vertical;
  width  = dvdq.horizontal - dvdp.horizontal;

  setColor(theGC,13); /* dark slate blue */
  if ((whichGadget == OVHEAD) || (whichGadget == ALT)) {
    XDrawRectangle(theDisplay,theWindow,theGC,dvdp.horizontal,dvdp.vertical,
      width,height);
  } else {
    XFillRectangle(theDisplay,theWindow,theGC,dvdp.horizontal,dvdp.vertical,
      width,height);
    dvdp.horizontal+=1;
    dvdp.vertical+=height;
    dvdp.vertical-=4;
    setColor(theGC,63); /* white */
    XDrawString(theDisplay,theWindow,theGC,dvdp.horizontal,
      dvdp.vertical,theString,strlen(theString));
  }/* end if */

}/* end function dvdDrawLine() */



/* Draw a line from p to q which has been presented in dvi coordinates. */
void dviDrawLine(Window theWindow,GC theGC,dviCoord dvip,dviCoord dviq)
{
  dvdCoord dvdp,dvdq;

  dvdp = dvi2dvd(dvip);
  dvdq = dvi2dvd(dviq);
  dvdDrawLine(theWindow,theGC,dvdp,dvdq);
}/* end function dvdDrawLine() */



/* Draw a line from p to q which has been presented in view-rectangle
** coordinates. */
void vrDrawLine(Window theWindow,GC theGC,vrCoord vrp,vrCoord vrq)
{
  dviCoord dvip,dviq;

  dvip = vr2dvi(vrp);
  dviq = vr2dvi(vrq);
  dviDrawLine(theWindow,theGC,dvip,dviq);
}/* end function vrDrawLine() */



/* Draw a line from p to q which has been presented in object-space
** coordinates. */
void osDrawLine(Window theWindow,GC theGC,osCoord osp,osCoord osq)
{
  vrCoord vrp,vrq;

  vrp = os2vr(osp);
  vrq = os2vr(osq);
  vrDrawLine(theWindow,theGC,vrp,vrq);
}/* end function osDrawLine() */
```

```
/* Set up the shadings needed for drawing shaded triangles, below. */
void initShading()
{
  int i;
  for (i = 0; i<= NUMSHADES-1; ++i) {
    theCoordParams.shadingColor[i] = i+NUMSHADES;
  }/* end for */
}/* end function initShading() */



/* Draw a triangle presented in object-space coordinates. */
void osDrawTriangle(Window theWindow,GC theGC,triangle theTriangle,
int theColor)
{
  osCoord osp,osq,osr;
  dvdCoord dvdp,dvdq,dvdr;
  XPoint   thePoints[4];

  osp.x = theTriangle.v1x; osp.y = theTriangle.v1y; osp.z = theTriangle.v1z;
  osq.x = theTriangle.v2x; osq.y = theTriangle.v2y; osq.z = theTriangle.v2z;
  osr.x = theTriangle.v3x; osr.y = theTriangle.v3y; osr.z = theTriangle.v3z;

  dvdp = dvi2dvd(vr2dvi(os2vr(osp)));
  dvdq = dvi2dvd(vr2dvi(os2vr(osq)));
  dvdr = dvi2dvd(vr2dvi(os2vr(osr)));

  thePoints[0].x = dvdp.horizontal; thePoints[0].y = dvdp.vertical;
  thePoints[1].x = dvdq.horizontal; thePoints[1].y = dvdq.vertical;
  thePoints[2].x = dvdr.horizontal; thePoints[2].y = dvdr.vertical;
  thePoints[3].x = dvdp.horizontal; thePoints[3].y = dvdp.vertical;

  switch(theCoordParams.solidType) {
  case WIREFRAME:
    setColor(theGC,theColor);
    XDrawLines(theDisplay,theWindow,theGC,thePoints,4,CoordModeOrigin);
    break;
  case HIDDEN:
    setColor(theGC,63);
    XFillPolygon(theDisplay,theWindow,theGC,thePoints,4,Convex,
      CoordModeOrigin);
    setColor(theGC,theColor);
    XDrawLines(theDisplay,theWindow,theGC,thePoints,4,CoordModeOrigin);
    break;
  case SHADED:
    setColor(theGC,theCoordParams.shadingColor[theColor]);
    XFillPolygon(theDisplay,theWindow,theGC,thePoints,4,Convex,
      CoordModeOrigin);
    /* setColor(theGC,1); black */
    XDrawLines(theDisplay,theWindow,theGC,thePoints,4,CoordModeOrigin);
    break;
  }/* end switch */

}/* end function osDrawTriangle() */



/* Draw a point p which has been presented in device-dependent
   coordinates. */
void dvdDrawPoint(Window theWindow,GC theGC,dvdCoord dvdp)
{
  XDrawPoint(theDisplay,theWindow,theGC,dvdp.horizontal,dvdp.vertical);
}/* end function dvdDrawPoint() */
```

```
** Draw a point p which has been presented in view-rectangle
** coordinates. */
void vrDrawPoint(Window theWindow,GC theGC,vrCoord vrp)
{
  dvdCoord dvdp;
  dviCoord dvip;

  dvip = vr2dvi(vrp);
  dvdp = dvi2dvd(dvip);
  XDrawPoint(theDisplay,theWindow,theGC,dvdp.horizontal,dvdp.vertical);
}/* end function vrDrawPoint() */



/* Draw a point p which has been presented in object-space coordinates. */
void osDrawPoint(Window theWindow,GC theGC,osCoord osp)
{
  vrCoord  vrp;

  vrp = os2vr(osp);
  vrDrawPoint(theWindow,theGC,vrp);
}/* end function osDrawPoint() */



** Render text with position presented in dvd coordinates. */
void dvdPrint(Window theWindow,GC theGC,dvdCoord dvdp,char theString[])
{
  XDrawString(theDisplay,theWindow,theGC,dvdp.horizontal,
    dvdp.vertical,theString,strlen(theString));
}/*  end function dvdPrint() */



/* Render text with position presented in vr coordinates. */
void vrPrint(Window theWindow,GC theGC,vrCoord vrp,char theString[])
{
  dvdCoord dvdp;
  dviCoord dvip;
  dvip = vr2dvi(vrp);
  dvdp = dvi2dvd(dvip);
  dvdPrint(theWindow,theGC,dvdp,theString);
}/*  end function vrPrint() */



** Render text with position presented in os coordinates. */
void osPrint(Window theWindow,GC theGC,osCoord osp,char theString[])
{
  vrCoord vrp;
  vrp = os2vr(osp);
  vrPrint(theWindow,theGC,vrp,theString);
/*  end function osPrint() */
```

```
/* The event handler. */

/* X-windows include files: */
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>
#include <X11/keysymdef.h>
#include <X11/cursorfont.h>
#include <math.h>

#include "global.h"
#include "prototypes.h"

/* Global X-Windows variables: */
extern Display *theDisplay;

/* global variables pertaining to this application: */
extern coordParams theCoordParams;
extern gadget      theGadgets[80];
extern int         drawGadgets;
extern int         nData;
extern int         highlight;
extern int         minDelta,maxDelta,numDelta;

extern int         nHull;
extern osCoord     theData[MAXDATA];
extern triangle    theHull[MAXTRIANGLES];
extern intTriangle intHull[MAXTRIANGLES];

/* Define the set of X-Windows events we wish to detect */
#define EV_MASK (KeyPressMask | ButtonPressMask | ExposureMask \
  | StructureNotifyMask)



  /* This is the main X-Windows event loop. Based on the events detected
  ** below (such as mouse clicks and key presses) various computations are
  ** performed, and then refreshWindow() is called.  If the event detected
  ** is a "quit" event, then control returns to the main program, where
  the X-Window is shut down and the program terminates. */
  int eventLoop()
  {
    XEvent             theEvent;
    XComposeStatus     theComposeStatus;
    KeySym             theKeySym;
    XWindowAttributes theAttribs;
    Cursor             theCursor;

    /* storage for incoming keystrokes */
    int  theKeyBufferMaxLen = 64; /* arbitrary big number */
    char theKeyBuffer[65];
    int  length;

    /* storage for mouse button events */
    int mouseX;
    int mouseY;
    int whichButton;

    /* other stuff */
    float sphereCoords[4];
    float tempCoords[4];
    int   i,whichGadget;
    float x,y,z;
```

```
/* wait for an event */
XNextEvent(theDisplay,&theEvent);

/* turn the cursor into a clock */
theCursor = XCreateFontCursor(theDisplay,XC_watch);
XDefineCursor(theDisplay,theEvent.xany.window,theCursor);
XFlush(theDisplay);

/* decode the event */
switch(theEvent.type) {

  /* the window has been exposed */
  case Expose:
    refreshWindow(theEvent.xany.window);
    break;

  /* the window is mapped */
  case MapNotify:
    refreshWindow(theEvent.xany.window);
    break;

  /* detect if a mouse button is pressed down */
  case ButtonPress:

    /* if drawGadgets equals false, then no button gadget could have
    ** been clicked into, so don't bother decoding this event */
    if (drawGadgets == False) { break; }

    /* figure out which button was pressed, and where on the screen it
    ** was pressed */
    mouseX = theEvent.xkey.x;
    mouseY = theEvent.xkey.y;
    whichButton = theEvent.xbutton.button;

    /* figure out which gadget was clicked into, if any */
    whichGadget = -1;
    for (i = QUIT; i <= NUMGADGETS; ++i) {
      if ((mouseX > theGadgets[i].left) && (mouseX < theGadgets[i].right)
      && (mouseY > theGadgets[i].top) && (mouseY < theGadgets[i].bottom)) {
        whichGadget = i;
      }/* end if */
    }/* end for */

    /* The ALT and OVHEAD gadgets are special; if the user did not click
    ** into one of them, then he might have clicked into one of the other
    ** movement gadgets.  If so, then we need to convert the coordinates
    ** he wants to change into spherical form so that we can move them as
    ** he requested, and then convert them back to rectangular coordinates.
    ** If the user _did_ click into the ALT or OVHEAD gadget, then he is
    ** moving one of the points directly, so we can keep them in
    ** rectangular coordinates.  */
    if ((whichGadget != ALT) && (whichGadget != OVHEAD)) {
      switch (whichButton) {
        case 1:
          tempCoords[1] = theCoordParams.eye[1] - theCoordParams.see[1];
          tempCoords[2] = theCoordParams.eye[2] - theCoordParams.see[2];
          tempCoords[3] = theCoordParams.eye[3] - theCoordParams.see[3];
          rect2sphere(tempCoords,sphereCoords); break;
        case 2: rect2sphere(theCoordParams.see,sphereCoords); break;
        case 3:
          tempCoords[1] = theCoordParams.light[1] - theCoordParams.see[1];
          tempCoords[2] = theCoordParams.light[2] - theCoordParams.see[2];
          tempCoords[3] = theCoordParams.light[3] - theCoordParams.see[3];
          rect2sphere(tempCoords,sphereCoords); break;
      }/* end switch */
    } else {
```

```
  switch (whichButton) {
    case 1: x = theCoordParams.eye[1]; y = theCoordParams.eye[2];
            z = theCoordParams.eye[3]; break;
    case 2: x = theCoordParams.see[1]; y = theCoordParams.see[2];
            z = theCoordParams.see[3]; break;
    case 3: x = theCoordParams.light[1]; y = theCoordParams.light[2];
            z = theCoordParams.light[3]; break;
  }/* end switch */
}/* end if */

/* now perform the various computations which depend on exactly which
** gadget was pressed */
switch (whichGadget) {
  case -1:      /* do nothing */                    break;
  case  QUIT:   return(0);                          break;
  case  WRITEDT: writeData();                       break;
  case  CCWISE:  sphereCoords[1]+=(1.0/36.0);       break;
  case  CLWISE:  sphereCoords[1]-=(1.0/36.0);       break;
  case  OVMAG:   theCoordParams.osLeft*=2.0;
                 theCoordParams.osBack*=2.0;
                 theCoordParams.osRight*=2.0;
                 theCoordParams.osFront*=2.0;
                 theCoordParams.osTop*=2.0;
                 theCoordParams.osBottom*=2.0; break;
  case  OVMIN:   theCoordParams.osLeft*=0.5;
                 if (theCoordParams.osLeft > theCoordParams.vcLeft) {
                   theCoordParams.osLeft*=2.0; break; }
                 theCoordParams.osBack*=0.5;
                 theCoordParams.osRight*=0.5;
                 theCoordParams.osFront*=0.5;
                 theCoordParams.osTop*=0.5;
                 theCoordParams.osBottom*=0.5; break;
  case  OVHEAD:  y = fromViewLeft(mouseX);
                 x = fromViewTop(mouseY);           break;
  case  ALT:     z = fromAltTop(mouseY);            break;
  case  ALTMAG:  theCoordParams.osLeft*=2.0;
                 theCoordParams.osBack*=2.0;
                 theCoordParams.osRight*=2.0;
                 theCoordParams.osFront*=2.0;
                 theCoordParams.osTop*=2.0;
                 theCoordParams.osBottom*=2.0; break;
  case  ALTMIN:  theCoordParams.osLeft*=0.5;
                 if (theCoordParams.osLeft > theCoordParams.vcLeft) {
                   theCoordParams.osLeft*=2.0; break; }
                 theCoordParams.osBack*=0.5;
                 theCoordParams.osRight*=0.5;
                 theCoordParams.osFront*=0.5;
                 theCoordParams.osTop*=0.5;
                 theCoordParams.osBottom*=0.5; break;
  case  INWARD:  sphereCoords[3]*=0.9;              break;
  case  OUTWARD: sphereCoords[3]*=1.1;              break;
  case  VIEWIN:  theCoordParams.rho+=0.05;
                 if (theCoordParams.rho > 0.95)
                   {theCoordParams.rho = 0.95;} break;
  case  VIEWOUT: theCoordParams.rho-=0.05;
                 if (theCoordParams.rho < 0.05)
                   {theCoordParams.rho = 0.05;} break;
  case  OVER:    sphereCoords[2]-=(1.0/36.0);       break;
  case  UNDER:   sphereCoords[2]+=(1.0/36.0);       break;
  case  AXESG:   theCoordParams.solidType = AXES;       break;
  case  WIREG:   theCoordParams.solidType = WIREFRAME;  break;
  case  HIDEG:   theCoordParams.solidType = HIDDEN;     break;
  case  SHADG:   theCoordParams.solidType = SHADED;     break;
  case  HIGHLT:  highlight = 1-highlight;               break;
  case  DOHULL:  chull();                               break;
  case  COARSE:  if (numDelta >   4) {
```

```
                    numDelta/=2;  nData = numDelta;
                    minDelta = 1; maxDelta = nData;
                    getData(); }                                  break;
  case  FINE:     if (numDelta < MAXDATA) {
                    numDelta*=2;  nData = numDelta;
                    minDelta = 1; maxDelta = nData;
                    getData(); }                                  break;
  case  MINDM:    minDelta--; nData = numDelta;
                  if (minDelta < 1) { minDelta = 1; }
                  getData();                                      break;
  case  MINDP:    minDelta++; nData = numDelta;
                  if (minDelta > maxDelta-3) { minDelta = maxDelta-3; }
                  getData();                                      break;
  case  MAXDM:    maxDelta--; nData = numDelta;
                  if (minDelta > maxDelta-3) { maxDelta = minDelta+3; }
                  getData();                                      break;
  case  MAXDP:    maxDelta++; nData = numDelta;
                  if (maxDelta > numDelta) { maxDelta = numDelta; }
                  getData();                                      break;
  case  X2:       theCoordParams.x_scale*=2;
                  theCoordParams.eye[1]*=2;
                  theCoordParams.see[1]*=2;
                  theCoordParams.light[1]*=2;                     break;
  case  X5:       theCoordParams.x_scale*=0.5;
                  theCoordParams.eye[1]*=0.5;
                  theCoordParams.see[1]*=0.5;
                  theCoordParams.light[1]*=0.5;                   break;
  case  Y2:       theCoordParams.y_scale*=2;
                  theCoordParams.eye[2]*=2;
                  theCoordParams.see[2]*=2;
                  theCoordParams.light[2]*=2;                     break;
  case  Y5:       theCoordParams.y_scale*=0.5;
                  theCoordParams.eye[2]*=0.5;
                  theCoordParams.see[2]*=0.5;
                  theCoordParams.light[2]*=0.5;                   break;
  case  Z2:       theCoordParams.z_scale*=2;
                  theCoordParams.eye[3]*=2;
                  theCoordParams.see[3]*=2;
                  theCoordParams.light[3]*=2;                     break;
  case  Z5:       theCoordParams.z_scale*=0.5;
                  theCoordParams.eye[3]*=0.5;
                  theCoordParams.see[3]*=0.5;
                  theCoordParams.light[3]*=0.5;                   break;
}/* end switch */

/* Now convert back to rectangular coordintes, if necessary. */
if ((whichGadget != ALT) && (whichGadget != OVHEAD)) {
  switch (whichButton) {
    case 1:
      sphere2rect(sphereCoords,tempCoords);
      theCoordParams.eye[1] = tempCoords[1] + theCoordParams.see[1];
      theCoordParams.eye[2] = tempCoords[2] + theCoordParams.see[2];
      theCoordParams.eye[3] = tempCoords[3] + theCoordParams.see[3];
      break;
    case 2: sphere2rect(sphereCoords,theCoordParams.see); break;
    case 3:
      sphere2rect(sphereCoords,tempCoords);
      theCoordParams.light[1] = tempCoords[1] + theCoordParams.see[1];
      theCoordParams.light[2] = tempCoords[2] + theCoordParams.see[2];
      theCoordParams.light[3] = tempCoords[3] + theCoordParams.see[3];
      break;
  }/* end switch */
} else {
  switch (whichButton) {
    case 1: theCoordParams.eye[1] = x; theCoordParams.eye[2] = y;
            theCoordParams.eye[3] = z; break;
```

```
        case 2: theCoordParams.see[1] = x; theCoordParams.see[2] = y;
                theCoordParams.see[3] = z; break;
        case 3: theCoordParams.light[1] = x; theCoordParams.light[2] = y;
                theCoordParams.light[3] = z; break;
      }/* end switch */
    }/* end if */

    /* Re-initialize the 3d->2D transformation and re-draw the window. */
    PSInit();
    refreshWindow(theEvent.xany.window);
    break;

  /* a key on the keyboard has been pressed down */
  case KeyPress:
    /* This line causes a compiler warning, but is legal, since theEvent */
    /* is of Union type. */
    length = XLookupString(&theEvent,theKeyBuffer,theKeyBufferMaxLen,
      &theKeySym,&theComposeStatus);

    /* check to see if the user pressed one of the ASCII keys */
    if ((theKeySym >= ' ') && (theKeySym <= '~') && (length == 1)) {

      /* quit if the key was a "q" */
      if (theKeyBuffer[0] == 'q') { return(0); }

      /* toggle the gadget drawing if the key was a "g" */
      if (theKeyBuffer[0] == 'g') drawGadgets = 1-drawGadgets;

      /* rotate the image quickly, if the key was a "a' */
      if (theKeyBuffer[0] == 'a') { /* animate! */
        for (i=1; i<=72; ++i) {
          tempCoords[1] = theCoordParams.eye[1] - theCoordParams.see[1];
          tempCoords[2] = theCoordParams.eye[2] - theCoordParams.see[2];
          tempCoords[3] = theCoordParams.eye[3] - theCoordParams.see[3];
          rect2sphere(tempCoords,sphereCoords);
          sphereCoords[1]+=(3.1415926/36.0);
          sphere2rect(sphereCoords,tempCoords);
          theCoordParams.eye[1] = tempCoords[1] + theCoordParams.see[1];
          theCoordParams.eye[2] = tempCoords[2] + theCoordParams.see[2];
          theCoordParams.eye[3] = tempCoords[3] + theCoordParams.see[3];
          PSInit();
          refreshWindow(theEvent.xany.window);
        }/* end for */
      }/* end if */

    /* the user must hav pressed a non-ASCII key */
    } else {
      /* as above, convert to spherical coordinates */
      tempCoords[1] = theCoordParams.eye[1] - theCoordParams.see[1];
      tempCoords[2] = theCoordParams.eye[2] - theCoordParams.see[2];
      tempCoords[3] = theCoordParams.eye[3] - theCoordParams.see[3];
      rect2sphere(tempCoords,sphereCoords);
      switch (theKeySym) {
        case XK_Up    : sphereCoords[2]-=(1.0/36.0); break;
        case XK_Down  : sphereCoords[2]+=(1.0/36.0); break;
        case XK_Right: sphereCoords[1]+=(1.0/36.0); break;
        case XK_Left  : sphereCoords[1]-=(1.0/36.0); break;
        case XK_KP_1  : sphereCoords[3]*=1.1;        break;
        case XK_KP_4  : sphereCoords[3]*=0.9;        break;
      }/* end switch */
      /* and then re-convert back to rectangular coordinates */
      sphere2rect(sphereCoords,tempCoords);
      theCoordParams.eye[1] = tempCoords[1] + theCoordParams.see[1];
      theCoordParams.eye[2] = tempCoords[2] + theCoordParams.see[2];
      theCoordParams.eye[3] = tempCoords[3] + theCoordParams.see[3];
      PSInit();
```

```
      }/* end if */
      refreshWindow(theEvent.xany.window);
      break;

    /* the window has been sized or changed */
    case ConfigureNotify:
      XGetWindowAttributes(theDisplay,theEvent.xany.window,&theAttribs);
      theCoordParams.windowWidth  = theAttribs.width;
      theCoordParams.windowHeight = theAttribs.height;
      refreshWindow(theEvent.xany.window);
      break;

  }/* end switch */

  /* make the pointer be an arrow again, instead of a clock */
  XUndefineCursor(theDisplay,theEvent.xany.window);

  return(1);
}/* end function eventLoop() */



/* function initEvents, set the event mask */
void initEvents(Window theWindow)
{
  XSelectInput(theDisplay,theWindow,EV_MASK);
}/* end function initEvents */
```

```
    /* PRINT SOME STATISTICS */

    dvdp.horizontal = theGadgets[X5].right + gadgetBorder;
    dvdp.vertical   = theGadgets[X5].bottom - 4;
    sprintf(theString,"scaling factor = %5.2f",theCoordParams.x_scale);
    dvdPrint(theExposedWindow,theGC,dvdp,theString);

    dvdp.horizontal = theGadgets[Y5].right + gadgetBorder;
    dvdp.vertical   = theGadgets[Y5].bottom - 4;
    sprintf(theString,"scaling factor = %5.2f",theCoordParams.y_scale);
    dvdPrint(theExposedWindow,theGC,dvdp,theString);

    dvdp.horizontal = theGadgets[Z5].right + gadgetBorder;
    dvdp.vertical   = theGadgets[Z5].bottom - 4;
    sprintf(theString,"scaling factor = %5.2f",theCoordParams.z_scale);
    dvdPrint(theExposedWindow,theGC,dvdp,theString);

    dvdp.horizontal = theGadgets[Z2].left;
    dvdp.vertical   = theGadgets[Z2].bottom + 3*gadgetBorder;
    sprintf(theString,"light point = (%7.2f,%7.2f,%7.2f)",
     theCoordParams.light[1],theCoordParams.light[2],theCoordParams.light[3]);
    dvdPrint(theExposedWindow,theGC,dvdp,theString);

    dvdp.vertical+=10;
    sprintf(theString,"  eye point = (%7.2f,%7.2f,%7.2f)",
     theCoordParams.eye[1],theCoordParams.eye[2],theCoordParams.eye[3]);
    dvdPrint(theExposedWindow,theGC,dvdp,theString);

    dvdp.vertical+=10;
    sprintf(theString,"  see point = (%7.2f,%7.2f,%7.2f)",
     theCoordParams.see[1],theCoordParams.see[2],theCoordParams.see[3]);
    dvdPrint(theExposedWindow,theGC,dvdp,theString);

    if ((highlight) && (theDepth > 1)) {
      dvdp.vertical+=30;
      setColor(theGC,49); /* red */
      sprintf(theString,"Red    - curve is exterior on far  side of hull");
      dvdPrint(theExposedWindow,theGC,dvdp,theString);
      dvdp.vertical+=10;
      setColor(theGC,1); /* black */
      sprintf(theString,"Yellow - curve is exterior on near side of hull");
      dvdPrint(theExposedWindow,theGC,dvdp,theString);
      dvdp.vertical+=10;
      setColor(theGC,4); /* brown */
      sprintf(theString,"Brown  - curve is in interior of hull");
      dvdPrint(theExposedWindow,theGC,dvdp,theString);
      dvdp.vertical+=10;
      setColor(theGC,1); /* black */
      sprintf(theString,"Green  - connecting line, if visible");
      dvdPrint(theExposedWindow,theGC,dvdp,theString);
    }/* end if */

  }/* end if(drawGadgets) */

  XFlush(theDisplay);
}/* end function refreshWindow() */



/* function to determine the shade of a facet of the hull */
int getShade(int i)
{
  float u[4];   /* a unit vector normal to the face of the hull */
  float v[4];   /* a unit vector pointing towards light source (lamp) */
  float w[4];   /* scratch storage */
```

```
  float angle;  /* angle between u and v */
  int   s;      /* shade number, based on angle */
  static float pi = 3.1415926;

  u[1] = theHull[i].normalx;
  u[2] = theHull[i].normaly;
  u[3] = theHull[i].normalz;
  w[1] = theCoordParams.light[1] - theHull[i].v3x;
  w[2] = theCoordParams.light[2] - theHull[i].v3y;
  w[3] = theCoordParams.light[3] - theHull[i].v3z;
  normalize(w,v);
  angle = acos(fabs(dotprd(u,v)));
  s = (NUMSHADES-1) - (int)(angle*2.0*(float)(NUMSHADES-1)/pi);
  return(s);
}/* end function getShade() */



/* function to sort the faces of the object */
void sortTriangles(int doDraw[],int sorted[])
{
  int   i,temp[MAXTRIANGLES];
  float diffx,diffy,diffz;
  float dist[MAXTRIANGLES];

  /* measure distances from eye points to triangles for all triangles */
  for (i = 1; i <= nHull+3; ++i) {
    diffx = theHull[i].centerx-theCoordParams.eye[1];
    diffy = theHull[i].centery-theCoordParams.eye[2];
    diffz = theHull[i].centerz-theCoordParams.eye[3];
    dist[i] = diffx*diffx + diffy*diffy + diffz*diffz;
  }/* end for */

  /* store the sorted list in a temporary variable */
  for (i = 1; i <= nHull+3; ++i) { temp[i] = i; }
  quickSort(dist,temp,1,nHull+3);

  /* reverse the order of the sort and return */
  for (i = 1; i <= nHull+3; ++i) { sorted[i] = temp[nHull+4-i]; }
}/* end function sortTriangles() */



int whichDraw(int doDraw[])
{
  int     i,counter;
  float   numer,sign1,sign2;
  float   normal[4],diff[4];

  counter = 0;
  for (i=1; i<=nHull; ++i) {

    /* project the center of the hull onto the normal of this triangle */
    normal[1] = theHull[i].normalx;
    normal[2] = theHull[i].normaly;
    normal[3] = theHull[i].normalz;
    diff[1] = centerHull.x - theHull[i].v3x;
    diff[2] = centerHull.y - theHull[i].v3y;
    diff[3] = centerHull.z - theHull[i].v3z;
    numer = dotprd(diff,normal);
    sign1 = numer;

    /* now project the eye point onto the normal of this triangle */
    diff[1] = theCoordParams.eye[1] - theHull[i].v3x;
```

```
    diff[2] = theCoordParams.eye[2] - theHull[i].v3y;
    diff[3] = theCoordParams.eye[3] - theHull[i].v3z;
    numer = dotprd(diff,normal);
    sign2 = numer;

    /* if sign2 and sign1 have the same sign, then our eye point is on the
    ** same side of the triangle as the the hull, and therefore we do
    ** NOT see this triangle. */
    if ((sign1*sign2) > 0.0) {
      doDraw[i] = 0;
    } else {
      doDraw[i] = 1;
      counter++;
    }/* end if */

  }/* end for i */

  /* CONSIDER THE AXES AS A SPECIAL CASE */
  /* the axes should always be drawn; rely on the sorting to put them in
  ** the right place in the rendering list so that they'll be obscured if
  ** they are behind the hull.  Note that there are some degenerate cases
  ** where this reliance is a mistake. */

  for (i=nHull+1; i<=nHull+3; ++i) {
    doDraw[i] = 1;
    counter++;
  }/* end for i */

  return(counter);
}/* end function whichDraw() */


/* figure out if edge connecting (p1,p2) is on triangle T or not */
int isAnEdge(int p1,int p2,intTriangle T)
{
  if (p1 == p2) { return(0); }
  if ((p1 == T.p1) || (p1 == T.p2) || (p1 == T.p3)) {
    if ((p2 == T.p1) || (p2 == T.p2) || (p2 == T.p3)) {
      return(1);
    }/* end if */
  }/* end if */
  return(0);
}/* end function isAnEdge() */
```

```
/* FILE: main.c
** This file contains the main program for the convex hull generator.  It also
** contains the refreshWindow() functions which draws the hull and gadgets. */

/* X-windows include files: */
#include <X11/Xlib.h>
#include <X11/Xutil.h>

/* include files for your application: */
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "global.h"          /* contains structure definitions and #define's */
#include "prototypes.h"      /* contains C prototypes */

/* X-windows global variables */
Display     *theDisplay;            /* a pointer to the display structure */
int          theScreen;             /* which screen within the display */
int          theDepth;              /* depth of screen in bitplanes */
unsigned     long theBlackPixel;    /* system black color, xet in colorx.c */
unsigned     long theWhitePixel;    /* system white color, set in colorx.c */
Colormap     theColormap;           /* color map, set in colorx.c */
GC           theGC;                 /* graphics context, set in windowx.c */
XFontStruct *fontStruct;            /* font structure, set in textx.c */

/* global variables pertaining to this application: */
osCoord     theData[MAXDATA+1];     /* the (x,y,z) triples of points */
int         nData;                  /* the number of such triples to be used */
triangle    theHull[MAXTRIANGLES+1];/* the triangles making up the hull */
intTriangle intHull[MAXTRIANGLES+1];/* data points on the curve mut the hull */
int         nHull;                  /* the number of triangles computed */
coordParams theCoordParams;         /* parameters related to image rendering */
Gadget      theGadgets[80];         /* gadget data */
int         drawGadgets;            /* whether or not gadgets should be drawn */
int         highlight;              /* whether or not curve should be highlit */
osCoord     centerHull;             /* a point in the center of convex hull */
int         minDelta,maxDelta,numDelta; /* see file getdata.c for explanation*/



void main()
{
  /* declarations pertaining to X-Windows: */
  Window       theWindow;
  int          windowX,windowY,popUp;
  int          windowWidth,windowHeight;

  /* declarations pertaining to this application: */
  /* (none) */

  /* initializations pertaining to X-Windows: */
  windowX = 0;              /* place the window in the upper left hand corner */
  windowY = 0;
  windowWidth  = -1;        /* -1 would mean make the window as big as possible */
  windowHeight = -1;
  popUp = 0;                /* an X parameter, see usage below */

  /* initializations pertaining to your application: */
  nData = 8;                /* use 8 data points for curve initially */
  minDelta = 1;             /* of those 8, use numbers 1 to 8 completely */
  maxDelta = nData;
  numDelta = maxDelta-minDelta+1;
  getData();     /* read in the input data file (this should be fixed) */
  setEye();      /* getData and setEye are, unfortunately, interdependent, so */
  getData();     /* read in the input data file a second time */
```

```
  theCoordParams.windowWidth    = windowWidth;
  theCoordParams.windowHeight   = windowHeight;
  theCoordParams.topBarHeight   = gadgetBorder*2+gadgetTall;
  theCoordParams.rightBarWidth  = gadgetBorder*7+gadgetWide*4+gadgetTall;
  theCoordParams.x_scale = 1.0;
  theCoordParams.y_scale = 1.0;
  theCoordParams.z_scale = 1.0;
  theCoordParams.rho    =  0.5;
  theCoordParams.tilt   =  0.0;
  theCoordParams.plotType  = MATHPLOT;     /* z=up, y=right, x=towards viewer */
  theCoordParams.solidType = AXES;         /* just draw curve & axes */
  drawGadgets = 1;                         /* do draw the gadgets */
  highlight   = 0;                         /* don't highlight (color) curve */
  PSInit();                                /* intialize 3D->2D projection */

  /* open up an X-Window: */
  initX();                          /* set up the connection to the X-Server */
  initDefaultColors();                        /* set up X-windows colors */
  initShading();                        /* initialize hull shading array */
  theWindow = openWindow(windowX,windowY,windowWidth,windowHeight,popUp,&theGC);
  fontStruct = initFont(theGC,"6x10");                    /* load in a font */
  initEvents(theWindow);                  /* set up window to receive events */

  refreshWindow(theWindow);            /* "refresh" window for the first time */
  XFlush(theDisplay);                  /* flush this refresh to the display */

  /* check for events; this is the heart of the program; most of the code
  ** would go here, or in the eventLoop() itself, or in the refreshWindow()
  ** function below: */
  while(eventLoop());                                      /* handle events */

  /* close everything down: */
  XDestroyWindow(theDisplay,theWindow);          /* free the window resources */
  XFreeFont(theDisplay,fontStruct);                /* free the font resources */
  XFlush(theDisplay);                               /* final flush to display */
  quitX();                                             /* exit from X-Windows */

}/* end main program */



  /* Function refreshWindow is probably the work-horse of the application.
  ** Based on events obtained from eventLoop(), you would perform various
  ** computations and then update the screen.  Most of that could probably
  ** happen in this routine. */
  void refreshWindow(Window theExposedWindow)
  {
    int      i,j;
    int      nDraw;                    /* how many triangle survive the culling */
    int      doDraw[MAXTRIANGLES];     /* which triangles survived the culling  */
    int      sorted[MAXTRIANGLES];     /* sort based on distance to eye point   */
    int      localShade[MAXTRIANGLES]; /* what color to shade each triangle     */
    int      colorBarWidth;            /* how wide the color bar should be      */
    char     theString[STRLEN];        /* generic string */
    dvdCoord dvdp,dvdq;                /* generic DVD coordinates */
    osCoord  osp,osq;                  /* generic OS coordinates */

    /* some computations that depend on how our window opened;
    ** how wide and high the convex hull display area is: */
    theCoordParams.dvdWidth  =
      theCoordParams.windowWidth  - theCoordParams.rightBarWidth*drawGadgets;
    theCoordParams.dvdHeight =
      theCoordParams.windowHeight - theCoordParams.topBarHeight*drawGadgets;
    colorBarWidth = (theCoordParams.dvdWidth-2*gadgetWide-3*gadgetBorder)/66;
```

```
/* clear the entire window */
XClearWindow(theDisplay,theExposedWindow);

/* DRAW THE OBJECT */

switch (theCoordParams.solidType) {
case AXES:                              /* draw just the curve and the axes */
  /* --- draw the axes --- */
  setColor(theGC,5);                    /* grey is the color of the axes */
  osp.x = 0.0 ; osp.y = 0.0 ; osp.z = 0.0;
  osPrint(theExposedWindow,theGC,osp,"o");
  osq.x = 0.5*theCoordParams.vcFront; osq.y = 0.0 ; osq.z = 0.0;
  osDrawLine(theExposedWindow,theGC,osp,osq);
  osPrint(theExposedWindow,theGC,osq,"x");
  osq.x = 0.0 ; osq.y = 0.5*theCoordParams.vcRight; osq.z = 0.0;
  osDrawLine(theExposedWindow,theGC,osp,osq);
  osPrint(theExposedWindow,theGC,osq,"y");
  osq.x = 0.0 ; osq.y = 0.0 ; osq.z = 0.5*theCoordParams.vcTop;
  osDrawLine(theExposedWindow,theGC,osp,osq);
  osPrint(theExposedWindow,theGC,osq,"z");
  /* --- draw the curve --- */
  setColor(theGC,1);                    /* black is the color of the curve */
  for (i=1; i <= nData-1; ++i) {
    osDrawLine(theExposedWindow,theGC,theData[i],theData[i+1]);
  }/* end for */
  break;
case WIREFRAME:                         /* draw the wireframe of the hull */
  /* --- draw the axes --- */
  setColor(theGC,5); /* grey */
  osp.x = 0.0 ; osp.y = 0.0 ; osp.z = 0.0;
  osPrint(theExposedWindow,theGC,osp,"o");
  osq.x = 0.5*theCoordParams.vcFront; osq.y = 0.0 ; osq.z = 0.0;
  osDrawLine(theExposedWindow,theGC,osp,osq);
  osPrint(theExposedWindow,theGC,osq,"x");
  osq.x = 0.0 ; osq.y = 0.5*theCoordParams.vcRight; osq.z = 0.0;
  osDrawLine(theExposedWindow,theGC,osp,osq);
  osPrint(theExposedWindow,theGC,osq,"y");
  osq.x = 0.0 ; osq.y = 0.0 ; osq.z = 0.5*theCoordParams.vcTop;
  osDrawLine(theExposedWindow,theGC,osp,osq);
  osPrint(theExposedWindow,theGC,osq,"z");
  /* --- draw the hull --- */
  for (i = 1; i <= nHull; ++i) {
    osDrawTriangle(theExposedWindow,theGC,theHull[i],1);
  }/* end for */
  break;
case HIDDEN:            /* draw the hull with hidden surface elimination */
  /* --- draw the axes labels --- */
  setColor(theGC,5); /* grey */
  osp.x = 0.0 ; osp.y = 0.0 ; osp.z = 0.0;
  osPrint(theExposedWindow,theGC,osp,"o");
  osq.x = 0.5*theCoordParams.vcFront; osq.y = 0.0 ; osq.z = 0.0;
  osPrint(theExposedWindow,theGC,osq,"x");
  osq.x = 0.0 ; osq.y = 0.5*theCoordParams.vcRight; osq.z = 0.0;
  osPrint(theExposedWindow,theGC,osq,"y");
  osq.x = 0.0 ; osq.y = 0.0 ; osq.z = 0.5*theCoordParams.vcTop;
  osPrint(theExposedWindow,theGC,osq,"z");

  /* --- draw the hull, and axes as triangles --- */
  /* figure out which of these triangles to draw, and set their color to
  ** black, so that they will have black outlines with white interior */
  nDraw = whichDraw(doDraw);
  for (i = 1; i <= nHull; ++i) {
    localShade[i] = 1; /* black */
  }/* end for */
  for (i = nHull+1; i <= nHull+3; ++i) { /* these triangles are the axes */
    localShade[i] = 5; /* grey */
```

```
  }/* end for */

  /* sort the triangles based on distance to the eye point */
  sortTriangles(doDraw,sorted);

  /* now draw the triangles */
  for (i = 1; i <= nHull+3; ++i) {
    if (doDraw[sorted[i]]) {
      osDrawTriangle(theExposedWindow,theGC,theHull[sorted[i]],
        localShade[sorted[i]]);
    }/* end if */
  }/* end for */
  break;

case SHADED:                          /* draw the hull with shaded facets */
  /* --- draw the axes labels --- */
  setColor(theGC,5); /* grey */
  osp.x = 0.0 ; osp.y = 0.0 ; osp.z = 0.0;
  osPrint(theExposedWindow,theGC,osp,"o");
  osq.x = 0.5*theCoordParams.vcFront; osq.y = 0.0 ; osq.z = 0.0;
  osPrint(theExposedWindow,theGC,osq,"x");
  osq.x = 0.0 ; osq.y = 0.5*theCoordParams.vcRight; osq.z = 0.0;
  osPrint(theExposedWindow,theGC,osq,"y");
  osq.x = 0.0 ; osq.y = 0.0 ; osq.z = 0.5*theCoordParams.vcTop;
  osPrint(theExposedWindow,theGC,osq,"z");

  /* --- draw the hull, and axes as triangles --- */
  /* figure out which of these triangles to draw, and set their color */
  nDraw = whichDraw(doDraw);
  for (i = 1; i <= nHull; ++i) {
    if (doDraw[i]) { localShade[i] = getShade(i); }
  }/* end for */
  for (i = nHull+1; i <= nHull+3; ++i) { /* these triangles are the axes */
    localShade[i] = 1;      /* 1 means black in my private shading system */
  }/* end for */

  /* sort the triangles based on distance to the eye point */
  sortTriangles(doDraw,sorted);

  /* now draw the triangles */
  for (i = 1; i <= nHull+3; ++i) {
    if (doDraw[sorted[i]]) {
      osDrawTriangle(theExposedWindow,theGC,theHull[sorted[i]],
        localShade[sorted[i]]);
    }/* end if */
  }/* end for */
  break;

}/* end switch */


if (highlight) {

  /* --- draw the curve in brown --- */
  if (theDepth == 1) { setColor(theGC,63); } /* white */
  else { setColor(theGC,4); } /* brown */
  for (i=1; i <= nData-1; ++i) {
    osDrawLine(theExposedWindow,theGC,theData[i],theData[i+1]);
  }/* end for */
  /* --- draw the invisible but exterior edges in red --- */
  if (theDepth == 1) { setColor(theGC,63); } /* white */
  else { setColor(theGC,49); } /* red */
  for (i=1; i <=nData-1; ++i) {
    for (j=1; j <= nHull; ++j) {
      if (isAnEdge(i,i+1,intHull[j])) {
        osDrawLine(theExposedWindow,theGC,theData[i],theData[i+1]);
```

```
        break;
      }/* end if */
    }/* end for j*/
  }/* end for i*/
  /* --- draw the exposed curve in yellow (white on a b&w monitor) --- */
  if (theDepth == 1) { setColor(theGC,63); } /* white */
  else { setColor(theGC,64); } /* yellow */
  nDraw = whichDraw(doDraw);
  for (i=1; i <=nData-1; ++i) {
    for (j=1; j <= nHull; ++j) {
      if (doDraw[j]) { /* this facet is being drawn */
        if (isAnEdge(i,i+1,intHull[j])) {
          osDrawLine(theExposedWindow,theGC,theData[i],theData[i+1]);
          break;
        }/* end if */
      }/* end if */
    }/* end for j*/
  }/* end for i*/

  /* draw the line connecting the two endpoints of the curve in green,
  ** assuming that it can be seen and is not obscured by the hull */
  if (theDepth == 1) { setColor(theGC,63); } /* white */
  else { setColor(theGC,21); } /* green */
  for (j=1; j <= nHull; ++j) {
    if (doDraw[j]) {
      if (isAnEdge(1,nData,intHull[j])) {
        osDrawLine(theExposedWindow,theGC,theData[1],theData[nData]);
        break;
      }/* end if */
    }/* end if */
  }/* end for j*/

}/* end if */


if (drawGadgets) {

/* clear the top and side bars */
XClearArea(theDisplay,theExposedWindow,0,0,0,theCoordParams.topBarHeight,
  False);
XClearArea(theDisplay,theExposedWindow,theCoordParams.dvdWidth,
  theCoordParams.topBarHeight,0,0,False);

  /* draw the borders of the top and right-bar gadget areas: */
  setColor(theGC,1);
  dvdp.horizontal = 0;
  dvdp.vertical   = theCoordParams.topBarHeight;
  dvdq.horizontal = theCoordParams.dvdWidth;
  dvdq.vertical   = theCoordParams.topBarHeight;
  dvdDrawLine(theExposedWindow,theGC,dvdp,dvdq);
  dvdp.horizontal = theCoordParams.dvdWidth;
  dvdp.vertical   = theCoordParams.topBarHeight;
  dvdq.horizontal = theCoordParams.dvdWidth;
  dvdq.vertical   = theCoordParams.windowHeight;
  dvdDrawLine(theExposedWindow,theGC,dvdp,dvdq);


  /* draw the gadgets themselves: */
  gadgetData();
  for (i = QUIT; i <= NUMGADGETS; ++i) {
    dvdDrawGadget(theExposedWindow,theGC,i);
  }/* end for */

  /* draw the color bars: */
  if (theDepth > 1) {
    /* --- 66 named colors --- */
```

```
ıorizontal = theCoordParams.dvdWidth-colorBarWidth;
ıorizontal = dvdp.horizontal+colorBarWidth;
rertical   = gadgetBorder;
rertical   = gadgetBorder+gadgetTall/2;
. = 0; i <= 65; ++i) {
:ol (theGC,i);
'illRectangle(theExposedWindow,theGC,dvdp,dvdq);
:olor(theGC,1);
!vdDrawRectangle(theExposedWindow,theGC,dvdp,dvdq); */
).horizontal-=colorBarWidth;
g.horizontal-=colorBarWidth;
:d for */
ıorizontal = theCoordParams.dvdWidth-colorBarWidth;
::rizontal = dvdp.horizontal+colorBarWidth;
rertical   = gadgetBorder+1+gadgetTall/2;
rertical   = gadgetBorder+gadgetTall;
- 66 shades of grey --- */
. = 66; i <= 131; ++i) {
:: or(theGC,i);
'..lRectangle(theExposedWindow,theGC,dvdp,dvdq);
:: or(theGC,1);
!- DrawRectangle(theExposedWindow,theGC,dvdp,dvdq); */
) orizontal-=colorBarWidth;
:- orizontal-=colorBarWidth;
i for */
. : */


pecial gadget items: */                                                    0);
                                                                           0);

HE VIEW-PLANE LOCATION DESCRIPTION: */
theGC,13); /* dark slate grey */
:al = theGadgets[VIEWIN].right;
:al   = theGadgets[VIEWIN].top   + 7;
izontal = theGadgets[VIEWOUT].left;
tical   = dvdp.vertical;
ine(theExposedWindow,theGC,dvdp,dvdq);
izontal+=
1.0-theCoordParams.rho)*(float)(dvdq.horizontal-dvdp.horizontal));
tical-=5;
izontal = dvdp.horizontal;
tical+=5;
ine(theExposedWindow,theGC,dvdp,dvdq);
izontal-=3;
tical+=8;
(theExposedWindow,theGC,dvdq,"v");


THE OVERHEAD VIEW DESCRIPTION */

rigin: */
(theGC,1); /* black */
izontal = (theGadgets[OVHEAD].left + theGadgets[OVHEAD].right)/2;
tical   = (theGadgets[OVHEAD].top + theGadgets[OVHEAD].bottom)/2;
oint(theExposedWindow,theGC,dvdp);

ye point: */
(theGC,4), /* brown */
izontal = viewLeft(theCoordParams.eye[2]);
tical   = viewTop(theCoordParams.eye[1]);
(th ExposedWindow,theGC,dvdp,"e");
oi theExposedWindow,theGC,dvdp);

ee point: */
(theGC,3); /* BlueViolet */
izontal = viewLeft(theCoordParams.see[2]);
```

```
dvdp.vertical+=4;
dvdPrint(theExposedWindow,theGC,dvdp,"s");

/* the light-point bar: */
setColor(theGC,18); /* Gold */
dvdp.horizontal = theGadgets[ALT].right-6;
dvdp.vertical   = altTop(theCoordParams.light[3]);
dvdq.horizontal = theGadgets[ALT].right;
dvdq.vertical   = dvdp.vertical;
dvdDrawLine(theExposedWindow,theGC,dvdp,dvdq);
dvdp.horizontal+=8;
dvdp.vertical+=4;
dvdPrint(theExposedWindow,theGC,dvdp,"l");

/* the top view-cube bar: */
setColor(theGC,5); /* cadet blue */
dvdp.horizontal = theGadgets[ALT].left;
dvdp.vertical   = altTop(theCoordParams.vcTop);
dvdq.horizontal = dvdp.horizontal+1;
dvdq.vertical   = dvdp.vertical;
dvdDrawLine(theExposedWindow,theGC,dvdp,dvdq);

/* the bottom view-cube bar: */
dvdp.horizontal = theGadgets[ALT].left;
dvdp.vertical   = altTop(theCoordParams.vcBottom);
dvdq.horizontal = dvdp.horizontal+1;
dvdq.vertical   = dvdp.vertical;
dvdDrawLine(theExposedWindow,theGC,dvdp,dvdq);

/* the view-cube tob/bottom connection bar: */
dvdp.horizontal = theGadgets[ALT].left+2;
dvdp.vertical   = altTop(theCoordParams.vcTop);
dvdq.horizontal = theGadgets[ALT].left+2;
dvdq.vertical   = altTop(theCoordParams.vcBottom);
dvdDrawLine(theExposedWindow,theGC,dvdp,dvdq);

/* the center bar: */
setColor(theGC,1); /* Black */
dvdp.horizontal = theGadgets[ALT].left+8;
dvdp.vertical   = (theGadgets[ALT].top + theGadgets[ALT].bottom)/2;
dvdq.horizontal = theGadgets[ALT].right-8;
dvdq.vertical   = dvdp.vertical;
dvdDrawLine(theExposedWindow,theGC,dvdp,dvdq);

/* DRAW THE NUMBER OF DATA AND HULL POINTS */

dvdp.horizontal = theGadgets[COARSE].right + gadgetBorder;
dvdp.vertical   = theGadgets[COARSE].top + 6;
sprintf(theString,"%3d points on curve",nData);
dvdPrint(theExposedWindow,theGC,dvdp,theString);

dvdp.vertical   = theGadgets[COARSE].bottom + 1;
sprintf(theString,"%3d facets on hull ",nHull);
dvdPrint(theExposedWindow,theGC,dvdp,theString);

/* DRAW THE LOWEST AND HIGHEST VALUES BEING USED */

dvdp.horizontal = theGadgets[MINDM].right + gadgetBorder;
dvdp.vertical   = theGadgets[MINDM].bottom - 4;
sprintf(theString," Using %d >= 1.",minDelta);
dvdPrint(theExposedWindow,theGC,dvdp,theString);

dvdp.vertical   = theGadgets[MAXDM].bottom - 4;
sprintf(theString," Using %d <= %d.",maxDelta,numDelta);
dvdPrint(theExposedWindow,theGC,dvdp,theString);
```

```
/* FILE: gadgetdata.c
** Routines to initialize gadget data. */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "global.h"
#include "prototypes.h"

extern coordParams theCoordParams;
extern gadget      theGadgets[80];
extern int         drawGadgets;



  /* initialize gadget data: where the gadget should be drawn and what
  ** string it should be labeled with */
  void gadgetData()
  {
    theGadgets[QUIT].top  = gadgetBorder;
    theGadgets[QUIT].bottom = theGadgets[QUIT].top + gadgetTall;
    theGadgets[QUIT].left = gadgetBorder;
    theGadgets[QUIT].right  = theGadgets[QUIT].left +gadgetWide;
    strcpy(theGadgets[QUIT].string,"Quit");

    theGadgets[WRITEDT].top  = theGadgets[QUIT].top;
    theGadgets[WRITEDT].bottom = theGadgets[QUIT].bottom;
    theGadgets[WRITEDT].left = theGadgets[QUIT].right + gadgetBorder;
    theGadgets[WRITEDT].right  = theGadgets[WRITEDT].left +gadgetWide;
    strcpy(theGadgets[WRITEDT].string,"Save Hull");

    /* observation-parameters control panel */

    theGadgets[CCWISE].top    = gadgetBorder;
    theGadgets[CCWISE].bottom = theGadgets[CCWISE].top + gadgetTall;
    theGadgets[CCWISE].left   = theCoordParams.dvdWidth + gadgetBorder;
    theGadgets[CCWISE].right  = theGadgets[CCWISE].left + gadgetWide;
    strcpy(theGadgets[CCWISE].string,"Counter C");

    theGadgets[OVMAG].top    = theGadgets[CCWISE].top;
    theGadgets[OVMAG].bottom = theGadgets[CCWISE].bottom;
    theGadgets[OVMAG].left   = theGadgets[CCWISE].right + gadgetBorder;
    theGadgets[OVMAG].right  = theGadgets[OVMAG].left + gadgetWide;
    strcpy(theGadgets[OVMAG].string,"Expand");

    theGadgets[OVMIN].top    = theGadgets[CCWISE].top;
    theGadgets[OVMIN].bottom = theGadgets[CCWISE].bottom;
    theGadgets[OVMIN].left   = theGadgets[OVMAG].right + gadgetBorder;
    theGadgets[OVMIN].right  = theGadgets[OVMIN].left + gadgetWide;
    strcpy(theGadgets[OVMIN].string,"Shrink");

    theGadgets[CLWISE].top    = theGadgets[CCWISE].top;
    theGadgets[CLWISE].bottom = theGadgets[CCWISE].bottom;
    theGadgets[CLWISE].left   = theGadgets[OVMIN].right + gadgetBorder;
    theGadgets[CLWISE].right  = theGadgets[CLWISE].left + gadgetWide;
    strcpy(theGadgets[CLWISE].string,"Clockwise");

    theGadgets[OVHEAD].top    = theGadgets[CCWISE].bottom + gadgetBorder;
    theGadgets[OVHEAD].bottom = theGadgets[OVHEAD].top
                              + (gadgetWide+gadgetBorder)*3;
    theGadgets[OVHEAD].left   = theGadgets[CCWISE].left;
    theGadgets[OVHEAD].right  = theGadgets[CLWISE].right;
    strcpy(theGadgets[OVHEAD].string,"");

    theGadgets[ALT].top    = theGadgets[OVHEAD].top;
    theGadgets[ALT].bottom = theGadgets[OVHEAD].bottom;
```

```
theGadgets[ALT].left   = theGadgets[OVHEAD].right + gadgetBorder;
theGadgets[ALT].right  = theGadgets[ALT].left + gadgetTall;
strcpy(theGadgets[ALT].string,"");

theGadgets[ALTMAG].top    = theGadgets[CCWISE].top;
theGadgets[ALTMAG].bottom = theGadgets[CCWISE].bottom;
theGadgets[ALTMAG].left   = theGadgets[ALT].left;
theGadgets[ALTMAG].right  = theGadgets[ALT].right;
strcpy(theGadgets[ALTMAG].string,"Ex");

theGadgets[INWARD].top    = theGadgets[OVHEAD].bottom + gadgetBorder;
theGadgets[INWARD].bottom = theGadgets[INWARD].top + gadgetTall;
theGadgets[INWARD].left   = theGadgets[CCWISE].left;
theGadgets[INWARD].right  = theGadgets[CCWISE].right;
strcpy(theGadgets[INWARD].string,"Zoom in");

theGadgets[OVER].top    = theGadgets[INWARD].top;
theGadgets[OVER].bottom = theGadgets[INWARD].bottom;
theGadgets[OVER].left   = theGadgets[OVMAG].left;
theGadgets[OVER].right  = theGadgets[OVMAG].right;
strcpy(theGadgets[OVER].string,"Go over");

theGadgets[UNDER].top    = theGadgets[INWARD].top;
theGadgets[UNDER].bottom = theGadgets[INWARD].bottom;
theGadgets[UNDER].left   = theGadgets[OVMIN].left;
theGadgets[UNDER].right  = theGadgets[OVMIN].right;
strcpy(theGadgets[UNDER].string,"Go under");

theGadgets[OUTWARD].top    = theGadgets[INWARD].top;
theGadgets[OUTWARD].bottom = theGadgets[INWARD].bottom;
theGadgets[OUTWARD].left   = theGadgets[CLWISE].left;
theGadgets[OUTWARD].right  = theGadgets[CLWISE].right;
strcpy(theGadgets[OUTWARD].string,"Zoom out");

theGadgets[ALTMIN].top    = theGadgets[INWARD].top;
theGadgets[ALTMIN].bottom = theGadgets[INWARD].bottom;
theGadgets[ALTMIN].left   = theGadgets[ALT].left;
theGadgets[ALTMIN].right  = theGadgets[ALT].right;
strcpy(theGadgets[ALTMIN].string,"Sh");

theGadgets[VIEWIN].top    = theGadgets[INWARD].bottom + gadgetBorder;
theGadgets[VIEWIN].bottom = theGadgets[VIEWIN].top + gadgetTall;
theGadgets[VIEWIN].left   = theGadgets[INWARD].left;
theGadgets[VIEWIN].right  = theGadgets[INWARD].right;
strcpy(theGadgets[VIEWIN].string,"See point");

theGadgets[VIEWOUT].top    = theGadgets[VIEWIN].top;
theGadgets[VIEWOUT].bottom = theGadgets[VIEWIN].bottom;
theGadgets[VIEWOUT].left   = theGadgets[OUTWARD].left;
theGadgets[VIEWOUT].right  = theGadgets[OUTWARD].right;
strcpy(theGadgets[VIEWOUT].string,"Eye point");

/* solid-type control panel */

theGadgets[AXESG].top    = theGadgets[VIEWIN].bottom + 5*gadgetBorder;
theGadgets[AXESG].bottom = theGadgets[AXESG].top + gadgetTall;
theGadgets[AXESG].left   = theGadgets[INWARD].left;
theGadgets[AXESG].right  = theGadgets[INWARD].right;
strcpy(theGadgets[AXESG].string,"Curve");

theGadgets[WIREG].top    = theGadgets[AXESG].top;
theGadgets[WIREG].bottom = theGadgets[AXESG].bottom;
theGadgets[WIREG].left   = theGadgets[OVER].left;
theGadgets[WIREG].right  = theGadgets[OVER].right;
strcpy(theGadgets[WIREG].string,"Wireframe");
```

```
theGadgets[HIDEG].top    = theGadgets[AXESG].top;
theGadgets[HIDEG].bottom = theGadgets[AXESG].bottom;
theGadgets[HIDEG].left   = theGadgets[UNDER].left;
theGadgets[HIDEG].right  = theGadgets[UNDER].right;
strcpy(theGadgets[HIDEG].string,"White");

theGadgets[SHADG].top    = theGadgets[AXESG].top;
theGadgets[SHADG].bottom = theGadgets[AXESG].bottom;
theGadgets[SHADG].left   = theGadgets[OUTWARD].left;
theGadgets[SHADG].right  = theGadgets[OUTWARD].right;
strcpy(theGadgets[SHADG].string,"Shaded");

theGadgets[HIGHLT].top    = theGadgets[AXESG].bottom + gadgetBorder;
theGadgets[HIGHLT].bottom = theGadgets[HIGHLT].top + gadgetTall;
theGadgets[HIGHLT].left   = theGadgets[SHADG].left;
theGadgets[HIGHLT].right  = theGadgets[SHADG].right;
strcpy(theGadgets[HIGHLT].string,"Highlight");

theGadgets[DOHULL].top    = theGadgets[HIGHLT].top;
theGadgets[DOHULL].bottom = theGadgets[HIGHLT].bottom;
theGadgets[DOHULL].left   = theGadgets[AXESG].left;
theGadgets[DOHULL].right  = theGadgets[WIREG].right;
strcpy(theGadgets[DOHULL].string,"COMPUTE CONVEX HULL");

/* grid refinement control panel */

theGadgets[COARSE].top    = theGadgets[HIGHLT].bottom + 5*gadgetBorder;
theGadgets[COARSE].bottom = theGadgets[COARSE].top + gadgetTall;
theGadgets[COARSE].left   = theGadgets[AXESG].left;
theGadgets[COARSE].right  = theGadgets[AXESG].right;
strcpy(theGadgets[COARSE].string,"Less fine");

theGadgets[FINE].top    = theGadgets[COARSE].top;
theGadgets[FINE].bottom = theGadgets[COARSE].bottom;
theGadgets[FINE].left   = theGadgets[SHADG].left;
theGadgets[FINE].right  = theGadgets[SHADG].right;
strcpy(theGadgets[FINE].string,"More fine");

/* parameter range control panel */

theGadgets[MINDM].top    = theGadgets[COARSE].bottom + 5*gadgetBorder;
theGadgets[MINDM].bottom = theGadgets[MINDM].top + gadgetTall;
theGadgets[MINDM].left   = theGadgets[COARSE].left;
theGadgets[MINDM].right  = theGadgets[COARSE].right;
strcpy(theGadgets[MINDM].string,"- t min");

theGadgets[MINDP].top    = theGadgets[MINDM].top;
theGadgets[MINDP].bottom = theGadgets[MINDM].bottom;
theGadgets[MINDP].left   = theGadgets[FINE].left;
theGadgets[MINDP].right  = theGadgets[FINE].right;
strcpy(theGadgets[MINDP].string,"+ t min");

theGadgets[MAXDM].top    = theGadgets[MINDM].bottom + gadgetBorder;
theGadgets[MAXDM].bottom = theGadgets[MAXDM].top + gadgetTall;
theGadgets[MAXDM].left   = theGadgets[MINDM].left;
theGadgets[MAXDM].right  = theGadgets[MINDM].right;
strcpy(theGadgets[MAXDM].string,"- t max");

theGadgets[MAXDP].top    = theGadgets[MAXDM].top;
theGadgets[MAXDP].bottom = theGadgets[MAXDM].bottom;
theGadgets[MAXDP].left   = theGadgets[MINDP].left;
theGadgets[MAXDP].right  = theGadgets[MINDP].right;
strcpy(theGadgets[MAXDP].string,"+ t max");

/* independent scaling of axes */
```

```
    theGadgets[X2].top    = theGadgets[MAXDM].bottom + 3 * gadgetBorder;
    theGadgets[X2].bottom = theGadgets[X2].top + gadgetTall;
    theGadgets[X2].left   = theGadgets[MAXDM].left;
    theGadgets[X2].right  = theGadgets[MAXDM].right;
    strcpy(theGadgets[X2].string," x * 2");

    theGadgets[X5].top    = theGadgets[X2].top;
    theGadgets[X5].bottom = theGadgets[X2].top + gadgetTall;
    theGadgets[X5].left   = theGadgets[X2].right + gadgetBorder;
    theGadgets[X5].right  = theGadgets[X5].left + gadgetWide;
    strcpy(theGadgets[X5].string," x / 2");

    theGadgets[Y2].top    = theGadgets[X2].bottom + gadgetBorder;
    theGadgets[Y2].bottom = theGadgets[Y2].top + gadgetTall;
    theGadgets[Y2].left   = theGadgets[X2].left;
    theGadgets[Y2].right  = theGadgets[X2].right;
    strcpy(theGadgets[Y2].string," y * 2");

    theGadgets[Y5].top    = theGadgets[Y2].top;
    theGadgets[Y5].bottom = theGadgets[Y2].top + gadgetTall;
    theGadgets[Y5].left   = theGadgets[X5].left;
    theGadgets[Y5].right  = theGadgets[X5].right;
    strcpy(theGadgets[Y5].string," y / 2");

    theGadgets[Z2].top    = theGadgets[Y2].bottom + gadgetBorder;
    theGadgets[Z2].bottom = theGadgets[Z2].top + gadgetTall;
    theGadgets[Z2].left   = theGadgets[Y2].left;
    theGadgets[Z2].right  = theGadgets[Y2].right;
    strcpy(theGadgets[Z2].string," z * 2");

    theGadgets[Z5].top    = theGadgets[Z2].top;
    theGadgets[Z5].bottom = theGadgets[Z2].top + gadgetTall;
    theGadgets[Z5].left   = theGadgets[Y5].left;
    theGadgets[Z5].right  = theGadgets[Y5].right;
    strcpy(theGadgets[Z5].string," z / 2");

}/* end function gadgetData() */
```

```
/* FILE: getdata.c
/* Routines to get the curve and hull data. */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <stdio.h>
#include <strings.h>
#include <math.h>
#include "global.h"
#include "prototypes.h"

extern osCoord     theData[MAXDATA];
extern int         nData;
extern triangle    theHull[MAXTRIANGLES];
extern intTriangle intHull[MAXTRIANGLES];
extern int         nHull;
extern coordParams theCoordParams;
extern osCoord     centerHull;
extern int         minDelta,maxDelta,numDelta;

/* The variables minDelta and maxDelta need some explanation.  Here goes:
** The data file which contains the (x,y,z) coordinate triples may have
** an arbitrary number of triples in it.  Typically, it's expected that there
** would be MAXDATA triples there, since that's the largest number of data point
** that this program is designed to use.  The variable nData would tell how
** many of those triples would be used.  For instance, if nData were set to
** 32, then every (MAXDATA/32)th triple would be used, and the others would be
** ignored.  Now, the user may also specify that he wants to chop off part
** of the curve: for instance the user might want to chop off the tail of
** the curve and view the convex hull of just the head.  In that case, we
** set minDelta to something larger than 1; for example, say minDelta = 3.
** Then of our 32 triples, the first and second would not be used.  Now,
** nData would have to be set to 30, since there are only 30 data points
** being used.  The variables numDelta would be set to 32, signifying that
** these 30 data points are a subset of the original 32.  If numDelta =
** nData, that means that the user wants to see all of the curve.  If the
** user wanted to also chop off the head of the curve, then maxDelta would
** be set to something less than 32.  Note that we always shift the data
** in the theData[] array so that the data we actually intend to use is
** stored in indices 1 through nData.  In this manner, all of this numDelta
** confusion is avoided in all of the other subroutines of this program. */


void getData()
{
  int   i,j,counter,k,ratio,nFile,nToUse,extra;
  float t,u[4],v[4],w[4];
  float centerx,centery,centerz;
  osCoord tempData;

  /* open the data file */
  FILE *f1, *fopen();
  if ((f1 = fopen("curve.data","r")) == NULL) {
    printf("Unable to open file %s for reading.\n","curve.data");
    exit(1);
  }

  /* check to make sure the data file has enough data in it */
  fscanf(f1,"%d",&nFile);
  if (nFile < 4) {
    printf("Insufficient data for generation of convex hull.\n");
    exit(1);
  }

  /* if the user asks for more points than there are in the data file
  ** then use all of the data in the data file; he'll have to be
```

```
** disappointed */
if (nFile < nData) {
  nData = nFile;
  printf
("%c You have requested more data than is available in the data file.\n",7);
}/* end if */

/* only read every ratio-th entry from the data file; note that we
** _always_ use the first and last data point in the file, so that the
** entire curve will be seen */
counter = 0;
ratio = nFile/nData;
extra = nFile - nData*ratio + ratio - 1;

/* read in the first nData-1 data points */
for (i = 1; i <= nData-1; ++i) {
  /* read this record and store it */
  fscanf(f1,"%f %f %f",&theData[i].x,&theData[i].y,&theData[i].z);
  counter++;
  for (j = 1; j <= ratio-1; ++j) {
    /* read this record and ignore it */
    fscanf(f1,"%f %f %f",&tempData.x,&tempData.y,&tempData.z);
    counter++;
  }/* end for j */
  if (i <= extra ) {
    /* read this extra record and ignore it */
    fscanf(f1,"%f %f %f",&tempData.x,&tempData.y,&tempData.z);
    counter++;
  }/* end if */
}/* end for i */

/* read in the nData'th data point as the last record in the file */
k = counter+1;
for (i = k; i <= nFile-1; ++i) {
  /* read this record and ignore it */
  fscanf(f1,"%f %f %f",&tempData.x,&tempData.y,&tempData.z);
  counter++;
}/* end for i */
i = nData;
/* read this record and store it */
fscanf(f1,"%f %f %f",&theData[i].x,&theData[i].y,&theData[i].z);
counter++;
fclose(f1);

/* if counter != nFile, then we screwed up somehow */
if (counter != nFile) {
  printf("Error reading data file: quantity of data is in error.\n");
}/* end if */

/* chop off the bottom or the top of the curve at the user's request */
nToUse = maxDelta-minDelta+1;
for (i=1; i <= nToUse; ++i) {
  theData[i] = theData[minDelta-1+i];
}/* end for */
numDelta = nData;
nData = nToUse;

/* compute the center of the curve */
centerx = 0.0; centery = 0.0; centerz = 0.0;
for (i = 1; i <= nData; ++i) {
  centerx = centerx + theData[i].x;
  centery = centery + theData[i].y;
  centerz = centerz + theData[i].z;
}/* end for */
centerx = centerx/(float)nData;
centery = centery/(float)nData;
```

```
centerz = centerz/(float)nData;
centerHull.x = centerx;
centerHull.y = centery;
centerHull.z = centerz;

nHull = 0;
/* Just as you would in the routine chull1...    */
/* ADD THREE MORE TRIANGLES: THE COORDINATE AXES */

/* I have a problem here: the values of vcFront, et al, are computed
** in the function setEye() below, and I don't want to call that function
** every time I call this function.  So really, the following code should
** be moved into its own function and called from main().  This is why
** getData() is currently called twice during the main initialization.
** Stupid mistake. */

/* the x-axis: */
theHull[nHull+1].v1x = 0.5*theCoordParams.vcFront;
theHull[nHull+1].v1y = 0.0;
theHull[nHull+1].v1z = 0.0;
theHull[nHull+1].v2x = 0.0; theHull[nHull+1].v3x = 0.0;
theHull[nHull+1].v2y = 0.0; theHull[nHull+1].v3y = 0.0;
theHull[nHull+1].v2z = 0.0; theHull[nHull+1].v3z = 0.0;

/* the y-axis: */
theHull[nHull+2].v1x = 0.0;
theHull[nHull+2].v1y = 0.5*theCoordParams.vcRight;
theHull[nHull+2].v1z = 0.0;
theHull[nHull+2].v2x = 0.0; theHull[nHull+2].v3x = 0.0;
theHull[nHull+2].v2y = 0.0; theHull[nHull+2].v3y = 0.0;
theHull[nHull+2].v2z = 0.0; theHull[nHull+2].v3z = 0.0;

/* the z-axis: */
theHull[nHull+3].v1x = 0.0;
theHull[nHull+3].v1y = 0.0;
theHull[nHull+3].v1z = 0.5*theCoordParams.vcTop;
theHull[nHull+3].v2x = 0.0; theHull[nHull+3].v3x = 0.0;
theHull[nHull+3].v2y = 0.0; theHull[nHull+3].v3y = 0.0;
theHull[nHull+3].v2z = 0.0; theHull[nHull+3].v3z = 0.0;

/* compute other aspects of the triangle data */
t = 0.333;
for (i = 1; i <= nHull+3; ++i) {

  /* compute the center of each triangle */
  theHull[i].centerx = (theHull[i].v1x + theHull[i].v2x + theHull[i].v3x)*t;
  theHull[i].centery = (theHull[i].v1y + theHull[i].v2y + theHull[i].v3y)*t;
  theHull[i].centerz = (theHull[i].v1z + theHull[i].v2z + theHull[i].v3z)*t;

  /* compute the normal of each triangle */
  u[1] = theHull[i].v1x - theHull[i].v3x;
  u[2] = theHull[i].v1y - theHull[i].v3y;
  u[3] = theHull[i].v1z - theHull[i].v3z;
  v[1] = theHull[i].v2x - theHull[i].v3x;
  v[2] = theHull[i].v2y - theHull[i].v3y;
  v[3] = theHull[i].v2z - theHull[i].v3z;
  cross(u,v,w); normalize(w,u);
  theHull[i].normalx = u[1];
  theHull[i].normaly = u[2];
  theHull[i].normalz = u[3];
/* end for */

/* the axes don't really have a unique normal, since they are lines */
for (i = nHull+1; i <= nHull+3; ++i) {
  theHull[i].normalx = centerx - theHull[i].v3x;
  theHull[i].normaly = centery - theHull[i].v3y;
```

```
    theHull[i].normalz = centerz - theHull[i].v3z;
  }/* end for */

}/* end function getData() */

/* COMPUTE THE CONVEX HULL */
void chull()
{
  int i;
  float t,u[4],v[4],w[4];
  float centerx,centery,centerz;

  nHull = hull3d(theData,theHull,intHull,nData);
  /* ADD THREE MORE TRIANGLES: THE COORDINATE AXES */

  /* the x-axis: */
  theHull[nHull+1].v1x = 0.5*theCoordParams.vcFront;
  theHull[nHull+1].v1y = 0.0;
  theHull[nHull+1].v1z = 0.0;
  theHull[nHull+1].v2x = 0.0; theHull[nHull+1].v3x = 0.0;
  theHull[nHull+1].v2y = 0.0; theHull[nHull+1].v3y = 0.0;
  theHull[nHull+1].v2z = 0.0; theHull[nHull+1].v3z = 0.0;

  /* the y-axis: */
  theHull[nHull+2].v1x = 0.0;
  theHull[nHull+2].v1y = 0.5*theCoordParams.vcRight;
  theHull[nHull+2].v1z = 0.0;
  theHull[nHull+2].v2x = 0.0; theHull[nHull+2].v3x = 0.0;
  theHull[nHull+2].v2y = 0.0; theHull[nHull+2].v3y = 0.0;
  theHull[nHull+2].v2z = 0.0; theHull[nHull+2].v3z = 0.0;

  /* the z-axis: */
  theHull[nHull+3].v1x = 0.0;
  theHull[nHull+3].v1y = 0.0;
  theHull[nHull+3].v1z = 0.5*theCoordParams.vcTop;
  theHull[nHull+3].v2x = 0.0; theHull[nHull+3].v3x = 0.0;
  theHull[nHull+3].v2y = 0.0; theHull[nHull+3].v3y = 0.0;
  theHull[nHull+3].v2z = 0.0; theHull[nHull+3].v3z = 0.0;

  /* compute other aspects of the triangle data */
  t = 0.333;
  for (i = 1; i <= nHull+3; ++i) {

    /* compute the center of each triangle */
    theHull[i].centerx = (theHull[i].v1x + theHull[i].v2x + theHull[i].v3x)*t;
    theHull[i].centery = (theHull[i].v1y + theHull[i].v2y + theHull[i].v3y)*t;
    theHull[i].centerz = (theHull[i].v1z + theHull[i].v2z + theHull[i].v3z)*t;

    /* compute the normal of each triangle */
    u[1] = theHull[i].v1x - theHull[i].v3x;
    u[2] = theHull[i].v1y - theHull[i].v3y;
    u[3] = theHull[i].v1z - theHull[i].v3z;
    v[1] = theHull[i].v2x - theHull[i].v3x;
    v[2] = theHull[i].v2y - theHull[i].v3y;
    v[3] = theHull[i].v2z - theHull[i].v3z;
    cross(u,v,w); normalize(w,u);
    theHull[i].normalx = u[1];
    theHull[i].normaly = u[2];
    theHull[i].normalz = u[3];
  }/* end for */

  /* the axes don't really have a unique normal, since they are lines */
```

```
  for (i = nHull+1; i <= nHull+3; ++i) {
    theHull[i].normalx = centerx - theHull[i].v3x;
    theHull[i].normaly = centery - theHull[i].v3y;
    theHull[i].normalz = centerz - theHull[i].v3z;
  }/* end for */
}/* end chull */


/* set the eye/see/light points */
void setEye()
{
  float l,r,T,b,F,B; /* left, right, Top, bottom, Front, Back */
  float maxall;
  int   i;

  /* compute maximum size needed for view cube */
  F = -1.0e32; r = F; T = F;
  B =  1.0e32; l = B; b = B;
  for (i = 1; i <= nData; ++i) {
    F = max2(F,theData[i].x);
    B = min2(B,theData[i].x);
    r = max2(r,theData[i].y);
    l = min2(l,theData[i].y);
    T = max2(T,theData[i].z);
    b = min2(b,theData[i].z);
  }/* end for */

  /* set the minimum/maximum initial values: */
  maxall = max4(fabs(F),fabs(B),fabs(r),fabs(l));
  maxall = max4(maxall,maxall,fabs(T),fabs(b));

  /* define the size of the view area, based on the size of the object
     we just read in */

  theCoordParams.vcRight  =  maxall;
  theCoordParams.vcLeft   = -maxall;
  theCoordParams.vcTop    =  maxall;
  theCoordParams.vcBottom = -maxall;
  theCoordParams.vcFront  =  maxall;
  theCoordParams.vcBack   = -maxall;

  theCoordParams.vrRight  =  0.4*maxall;
  theCoordParams.vrLeft   = -0.4*maxall;
  theCoordParams.vrTop    =  0.4*maxall;
  theCoordParams.vrBottom = -0.4*maxall;

  theCoordParams.osRight  =  20.0*maxall;
  theCoordParams.osLeft   = -20.0*maxall;
  theCoordParams.osTop    =  20.0*maxall;
  theCoordParams.osBottom = -20.0*maxall;
  theCoordParams.osFront  =  20.0*maxall;
  theCoordParams.osBack   = -20.0*maxall;

  theCoordParams.eye[1]   =  10.0*maxall;
  theCoordParams.eye[2]   =   5.0*maxall;
  theCoordParams.eye[3]   =   2.0*maxall;

  theCoordParams.light[1]   = 10.0*maxall;
  theCoordParams.light[2]   = 10.0*maxall;
  theCoordParams.light[3]   = 10.0*maxall;

  theCoordParams.see[1]   =  (F+B)*0.5;
  theCoordParams.see[2]   =  (r+1)*0.5;
  theCoordParams.see[3]   =  (T+b)*0.5;
```

```
}/* end function setEye() */


/* write out the convex hull data to a file, at the user's request */
void writeData()
{
  int i;

  FILE *fl, *fopen();

  if ((fl = fopen("hull.data","w")) == NULL) {
    printf("Unable to open file %s for writing.\n","hull.data");
    exit(1);
  }/* end if */

  fprintf(fl,"%d\n",nHull);
  for (i=1; i <= nHull; ++i) {
    fprintf(fl,"%f %f %f   %f %f %f   %f %f %f\n",
      theHull[i].v1x,theHull[i].v1y,theHull[i].v1z,
      theHull[i].v2x,theHull[i].v2y,theHull[i].v2z,
      theHull[i].v3x,theHull[i].v3y,theHull[i].v3z);
  }/* end for */

  fclose(fl);

}/* end function write() */
```

```
/* FILE: intix.c
** Initialize the connection to the X-server. */

/* X-windows include files: */
#include <X11/Xlib.h>
#include <X11/Xutil.h>

/* Standard I/O include file: */
#include <stdio.h>

#include "global.h"
#include "prototypes.h"

/* Global variables: */
extern Display *theDisplay;
extern int      theScreen;
extern int      theDepth;
extern unsigned long theBlackPixel;
extern unsigned long theWhitePixel;
extern Colormap theColormap;



  /* Function initX sets up the connection to the X-server and stores
  ** information about the environent. */
  void initX()
  {
    /* Establish a connection to the X-server: */
    theDisplay = XOpenDisplay(NULL);

    /* Check to make sure the display opened okay: */
    if (theDisplay == NULL) {
      fprintf(stderr,
        "ERROR: Cannot establish a connection to the X-Server %s\n",
        XDisplayName(NULL));
      exit(1);
    }/* end if */

    /* Find out what the default screen and it's (color) depth is.
    ** If theDepth == 1 then we have a monochrome system. */
    theScreen = DefaultScreen(theDisplay);
    theDepth  = DefaultDepth(theDisplay,theScreen);
    theBlackPixel = BlackPixel(theDisplay,theScreen);
    theWhitePixel = WhitePixel(theDisplay,theScreen);
    theColormap   = DefaultColormap(theDisplay,theScreen);

  }/* end function initX() */



  /* Function getXinfo prints out information about the current X-Window
  ** display and screen.  Entirely optional to include this, of course.  */
  void getXInfo()
  {
    printf("%s version %d of the X Window System, X%d R%d\n",
      ServerVendor(theDisplay),
      VendorRelease(theDisplay),
      ProtocolVersion(theDisplay),
      ProtocolRevision(theDisplay));

    if (theDepth == 1) {
      printf("Color plane depth.....%d (monochrome)\n",theDepth);
    } else {
      printf("Color plane depth.....%d\n",theDepth);
```

```
  }/* end if */

  printf("Display width.........%d\n",DisplayWidth(theDisplay,theScreen));
  printf("Display height........%d\n",DisplayHeight(theDisplay,theScreen));
  /* printf("The display %s\n",XDisplayName(theDisplay)); */

}/* end function getXInfo() */
```

```
/* FILE: hull.c
** Function to compute the convex hull of a set of 3D points.
** (Programming note: in this file the left-handed cross product function
** from the 3d.c file is used.  Accordingly, the sign is changed here to
** make it a right-handed cross product.) */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <stdio.h>
#include <math.h>
#include "global.h"
#include "prototypes.h"



/* Returns the number of points in the hull. */
int hull3d(osCoord theData[],triangle theHull[],intTriangle intHull[],int n)
{
  int i,nHull;
  nHull = GiftWrapping(theData,theHull,intHull,n);
  return(nHull);
}/* end function hull2d() */




/* ------------------------------------------------------------------------*/
/* The giftwrapping algorithm, taken from page 128 of Preparata and Shamos. */
/* Note that I have modified the algorithm somewhat.  Their version had     */
/* some bugs.                                                               */

/* typedef struct { int vertex[4]; } Triangle; */
/* typedef struct { int vertex[3]; } Edge;     */



/* This function returns the (integer) number of triangles in the hull. */
int GiftWrapping(osCoord theData[],triangle theHull[],intTriangle intHull[],int
{
  int      i,numH,numQ,numT;
  int      freq[MAXDATA];   /* counts how often each point is used */
  Triangle F,Fprime;        /* F is a single facet */
  Triangle Q[MAXTRIANGLES]; /* Q is a queue of facets */
  Edge     T[MAXEDGES];     /* T is a file of edges */
  Edge     t[4];            /* t is a list of the edges in F */
  Edge     e;               /* e is a single edge */

  /* set frequency count to zero; used for 2D degeneracies where the same
  ** point might get chosen over and over again, if we didn't try to choose
  ** the least-frequently-used points */
  for (i=1; i<=MAXDATA-1; ++i) { freq[i] = 0; }

  /* Q := empty set */
  numQ = 0;

  /* T := empty set */
  numT = 0;

  /* find an initial starting facet */
  numH = 0;
  F = findFirstFacet(theData,theHull,freq,n);
  numH = storeh(F,theHull,intHull,theData,numH);

  /* T <== subfacets of F */
```

```
  numT = addToFile(F,T,numT);

  /* Q <== F */
  numQ = push(F,Q,numQ);

  while (numQ > 0) {  /* while Q != empty set */
    F = pop(Q,numQ); numQ--;
    getEdges(F,t);
    /* for each e in t intersect T */
    for (i=1; i<=3; ++i ) {
      if (isCommon(t[i],T,numT)) {
        e.vertex[1] = t[i].vertex[1];
        e.vertex[2] = t[i].vertex[2];
        Fprime = giftWrap(e,F,theData,freq,n);
        numT = insertDelete(Fprime,T,numT);
        if (numT > MAXEDGES-1) {
          printf("%c ERROR: Insufficient workspace for computing edges.\n",7);
          return(numH);
        }/* end if */
        numQ = push(Fprime,Q,numQ);
        if (numQ > MAXTRIANGLES-1) {
          printf("%c ERROR: Insufficient workspace for computing hull.\n",7);
          return(numH);
        }/* end if */
      }/* end if */
    }/* end for each */
    numH = storeh(F,theHull,intHull,theData,numH);
    if (numH > MAXTRIANGLES-4) {
      printf("%c ERROR: Hull consists of too many triangles for the storage allo
      return(numH);
    }/* end if */
  }/* end while */

  numH--;
  return(numH);
}/* end function GiftWrapping() */



/* Very loosely based on page 129 of Preparata and Shamos. */
Triangle findFirstFacet(osCoord theData[],triangle theHull[],int freq[],
int numD)
{
  int       i,p1,p2,p3;
  float     a[4],n[4],x[4],y[4],z[4],u[4],v[4],w[4];
  float     minz,extRho,rho;
  Triangle theTriangle;

  /* find the lowest point among all the data */
  minz = theData[1].z; p1 = 1;
  for (i=2; i<=numD; ++i) {
    if (theData[i].z < minz) { minz = theData[i].z; p1 = i; }
  }/* end for */

  /* iteration #2:
  */
    /* x is in the direction of the x-axis */
    x[1] = 1.0; x[2] = 0.0; x[3] = 0.0;

    /* y is in the direction of the y-axis */
    y[1] = 0.0; y[2] = 1.0; y[3] = 0.0;

    /* z is in the direction of the z-axis */
    z[1] = 0.0; z[2] = 0.0; z[3] = 1.0;
```

```
    extRho =  1.0e32; p2 = 0;
    for (i=1; i<=numD; ++i) {
      if (i == p1) { continue; }

      /* v is the direction of the proposed first edge */
      w[1] = theData[i].x - theData[p1].x;
      w[2] = theData[i].y - theData[p1].y;
      w[3] = theData[i].z - theData[p1].z;
      normalize(w,v);

      /* we want the edge which is 'most obtuse' to the x-axis */
      rho  = dotprd(v,x);
      if (rho < extRho) {
        extRho = rho;
        p2 = i;
      }/* end if */

    }/* end for */
    if (p2 == 0) {
      printf("%c ERROR: Unable to find second starting point.\n",7);
      exit(1);
     }/* end if */

  /* iteration #3:
  */
    /* u is the direction of the edge we found above */
    w[1] = theData[p2].x - theData[p1].x;
    w[2] = theData[p2].y - theData[p1].y;
    w[3] = theData[p2].z - theData[p1].z;
    normalize(w,u);

    extRho =  1.0e32; p3 = 0;
    for (i=1; i<=numD; ++i) {
      if ((i == p1) || (i == p2)) { continue; }

      /* v is the direction of the proposed second edge */
      w[1] = theData[i].x - theData[p1].x;
      w[2] = theData[i].y - theData[p1].y;
      w[3] = theData[i].z - theData[p1].z;
      normalize(w,v);

      /* n is the normal of the proposed first face */
      cross(u,v,a); normalize(a,n);

      /* we want the normal to be 'most obtuse' to the x-axis */
      rho  = dotprd(n,x);
      if (rho < extRho) {
        extRho = rho;
        p3 = i;
      }/* end if */

    }/* end for */
    if (p3 == 0) {
      printf("%c ERROR: Unable to find third starting point.\n",7);
      exit(1);
     }/* end if */

  theTriangle.vertex[1] = p1;
  theTriangle.vertex[2] = p2;
  theTriangle.vertex[3] = p3;
  freq[p1]++; freq[p2]++; freq[p3]++;
  return(theTriangle);
}
```

```
/* only add F to the T file if it's not already there */
int addToFile(Triangle F,Edge T[],int numT)
{
  int numV1,numV2,a;
  int i,j;
  for (i=1; i<=3; ++i) {
    if (i==1) { numV1 = F.vertex[1]; numV2 = F.vertex[2]; }
    if (i==2) { numV1 = F.vertex[2]; numV2 = F.vertex[3]; }
    if (i==3) { numV1 = F.vertex[3]; numV2 = F.vertex[1]; }
    a = False;
    for (j=1; j<=numT; ++j) {
      if (((numV1 == T[j].vertex[1]) && (numV2 == T[j].vertex[2]))
       || ((numV1 == T[j].vertex[2]) && (numV2 == T[j].vertex[1])))
        { a=True; }
    }/* end for j */
    if (a == False) {
      numT++;
      T[numT].vertex[1] = numV1;
      T[numT].vertex[2] = numV2;
    }/* end if */
  }/* end for i */
  return(numT);
}


/* push a face onto the stack ONLY if it's not already there */
int push(Triangle F,Triangle Q[],int numQ)
{
  int  i,faceIsNew;
  int  s[4],t[4];

  faceIsNew = True;
  s[1] = F.vertex[1]; s[2] = F.vertex[2]; s[3] = F.vertex[3];
  for (i=1; i<= numQ; ++i) {
    t[1] = Q[i].vertex[1]; t[2] = Q[i].vertex[2]; t[3] = Q[i].vertex[3];
    if
     ( ((s[1]==t[1]) && (s[2]==t[2]) && (s[3]==t[3]))
    || ((s[1]==t[1]) && (s[3]==t[2]) && (s[2]==t[3]))
    || ((s[2]==t[1]) && (s[1]==t[2]) && (s[3]==t[3]))
    || ((s[2]==t[1]) && (s[3]==t[2]) && (s[1]==t[3]))
    || ((s[3]==t[1]) && (s[1]==t[2]) && (s[2]==t[3]))
    || ((s[3]==t[1]) && (s[2]==t[2]) && (s[1]==t[3]))
     ) { faceIsNew = False; }
  }/* end for */

  if (faceIsNew) {
    numQ++;
    Q[numQ].vertex[1] = F.vertex[1];
    Q[numQ].vertex[2] = F.vertex[2];
    Q[numQ].vertex[3] = F.vertex[3];
  }/* end if */
  return(numQ);
}



/* pop a face off the stack */
Triangle pop(Triangle Q[],int numQ)
{
  Triangle F;
  F.vertex[1] = Q[numQ].vertex[1];
  F.vertex[2] = Q[numQ].vertex[2];
  F.vertex[3] = Q[numQ].vertex[3];
  return(F);
}
```

```
/* find out which edges belong to face F */
void getEdges(Triangle F,Edge t[])
{
  t[1].vertex[1] = F.vertex[1];
  t[1].vertex[2] = F.vertex[2];

  t[2].vertex[1] = F.vertex[2];
  t[2].vertex[2] = F.vertex[3];

  t[3].vertex[1] = F.vertex[1];
  t[3].vertex[2] = F.vertex[3];
}



/* see if edge at is in the list of edges T */
int isCommon(Edge at,Edge T[],int numT)
{
  int j,a;
  a = False;
  for (j=1; j <= numT; ++j) {
    if (((at.vertex[1] == T[j].vertex[1]) && (at.vertex[2] == T[j].vertex[2]))
     || ((at.vertex[1] == T[j].vertex[2]) && (at.vertex[2] == T[j].vertex[1])))
     { a=True; }
  }/* end for j */
  return(a);
}



/* very loosely based on page 127 of Preparata and Shamos. */
Triangle giftWrap(Edge e,Triangle F,osCoord theData[],int freq[],int numD)
{
  int      i,k;
  osCoord  p1,p2,p3;
  float    u[4],v[4],w[4],n[4],E[4],a[4],rho,extRho;
  Triangle newF;

  /* find out which point of face F is not in edge e */
  for (i=1; i <=3 ; ++i) {
    if ((F.vertex[i] != e.vertex[1]) && (F.vertex[i] != e.vertex[2]))
    { k = i; }
  }/* end for */

  p1.x = theData[e.vertex[1]].x;
  p1.y = theData[e.vertex[1]].y;
  p1.z = theData[e.vertex[1]].z;

  p2.x = theData[e.vertex[2]].x;
  p2.y = theData[e.vertex[2]].y;
  p2.z = theData[e.vertex[2]].z;

  p3.x = theData[F.vertex[k]].x;
  p3.y = theData[F.vertex[k]].y;
  p3.z = theData[F.vertex[k]].z;

  /* E is one edge of the face we are wrapping around */
  E[1] = p2.x-p1.x; E[2] = p2.y-p1.y; E[3] = p2.z-p1.z;

  /* n is normal to the face we are wrapping around */
  v[1] = p3.x-p1.x; v[2] = p3.y-p1.y; v[3] = p3.z-p1.z;
  cross(E,v,w); normalize(w,n);
  n[1]*=(-1.0); n[2]*=(-1.0); n[3]*=(-1.0);
```

```
extRho =  1.0e32; i=0;
for (k=1; k<=numD; ++k) {
  if ((k==F.vertex[1]) || (k==F.vertex[2]) || (k==F.vertex[3])) { continue; }
  w[1] = theData[k].x-p1.x;
  w[2] = theData[k].y-p1.y;
  w[3] = theData[k].z-p1.z;
  normalize(w,v);

  /* a is normal to both E and v, which are on the proposed new face */
  cross(E,v,w); normalize(w,a);
  a[1]*=(-1.0); a[2]*=(-1.0); a[3]*=(-1.0);

  /* we want the proposed new face to be 'most obtuse' with the old face */
  rho = dotprd(n,a);
  if (rho < extRho) {
    extRho = rho;
    i = k;
  } else {
    /* in case of a tie, choose the point least frequently used; this
    ** ensures that as many points 'on the curve' as possible will be
    ** used in the convex hull --- this is important for 2D degeneracies */
    if (rho == extRho) {
      if (freq[k] < freq[i]) {
        i = k;
      }/* end if */
    }/* end if */
  }/* end if-else */

}/* end for */
if (i == 0) {
  printf("%c ERROR: Unable to giftwrap.  Degenerate problem?\n",7);
  exit(1);
}/* end if */

newF.vertex[1] = e.vertex[1];
newF.vertex[2] = e.vertex[2];
newF.vertex[3] = i;
freq[i]++;
return(newF);
```

```
/* if an edge of face F is in the list T, then delete it; else add it */
int insertDelete(Triangle F,Edge T[],int numT)

  int  numV1,numV2,a;
  int  counter,filled[MAXEDGES];
  Edge tempT[MAXEDGES];
  int  i,j;

  for (i=1; i<=numT; ++i) { filled[i] = True; }
  for (i=1; i<=3; ++i) {
    if (i==1) { numV1 = F.vertex[1]; numV2 = F.vertex[2]; }
    if (i==2) { numV1 = F.vertex[2]; numV2 = F.vertex[3]; }
    if (i==3) { numV1 = F.vertex[3]; numV2 = F.vertex[1]; }
    a = False;
    for (j=1; j<=numT; ++j) {
      if (filled[j] == False) { continue; }
      if (((numV1 == T[j].vertex[1]) && (numV2 == T[j].vertex[2]))
       || ((numV1 == T[j].vertex[2]) && (numV2 == T[j].vertex[1])))
       { a=True; filled[j] = False; /* delete this element from the list */}
    }/* end for j */
    if (a == False) { /* add this element to the list */
      numT++;
      filled[numT] = True;
```

```
      T[numT].vertex[1] = numV1;
      T[numT].vertex[2] = numV2;
    }/* end if */
  }/* end for i */

  re-compress the list */
  counter = 0;
  for (i=1; i<=numT; ++i) {
    if (filled[i]) { counter++; tempT[counter] = T[i]; }
  }/* end for */
  for (i=1; i<=counter; ++i) {
    T[i] = tempT[i];
  }/* end for */

  return(counter);
}



/* store the triangle F as a set of 9 floating points numbers (three
** vertices) as well as 3 integers (the points on the curve) */
int storeh(Triangle F,triangle theHull[],intTriangle intHull[],osCoord theData[]
{
  theHull[n].v1x = theData[F.vertex[1]].x;
  theHull[n].v1y = theData[F.vertex[1]].y;
  theHull[n].v1z = theData[F.vertex[1]].z;

  theHull[n].v2x = theData[F.vertex[2]].x;
  theHull[n].v2y = theData[F.vertex[2]].y;
  theHull[n].v2z = theData[F.vertex[2]].z;

  theHull[n].v3x = theData[F.vertex[3]].x;
  theHull[n].v3y = theData[F.vertex[3]].y;
  theHull[n].v3z = theData[F.vertex[3]].z;

  intHull[n].p1 = F.vertex[1];
  intHull[n].p2 = F.vertex[2];
  intHull[n].p3 = F.vertex[3];

  n++;
  return(n);
}
```

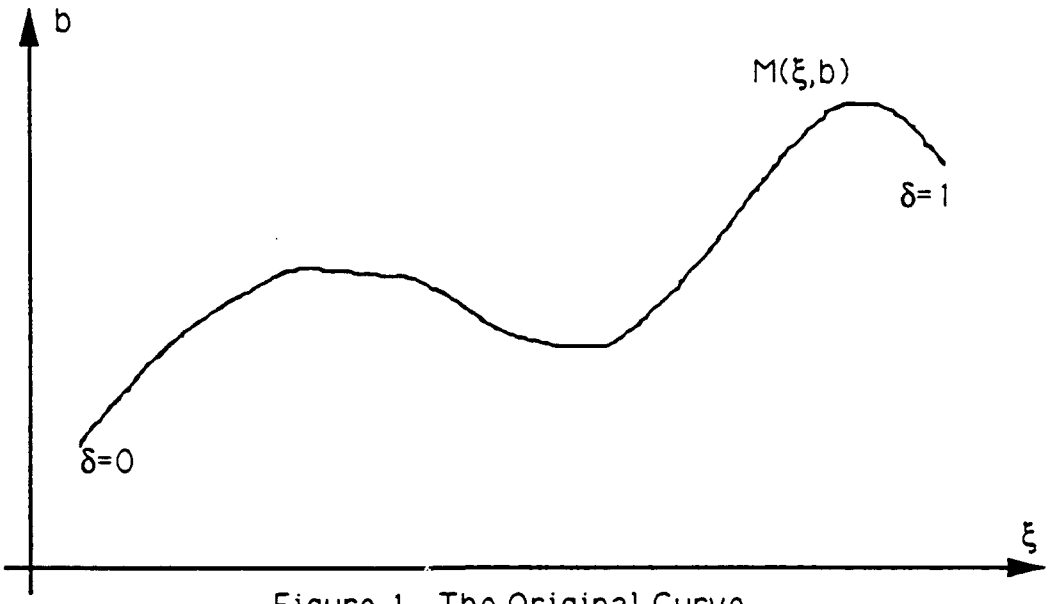


Figure 1. The Original Curve

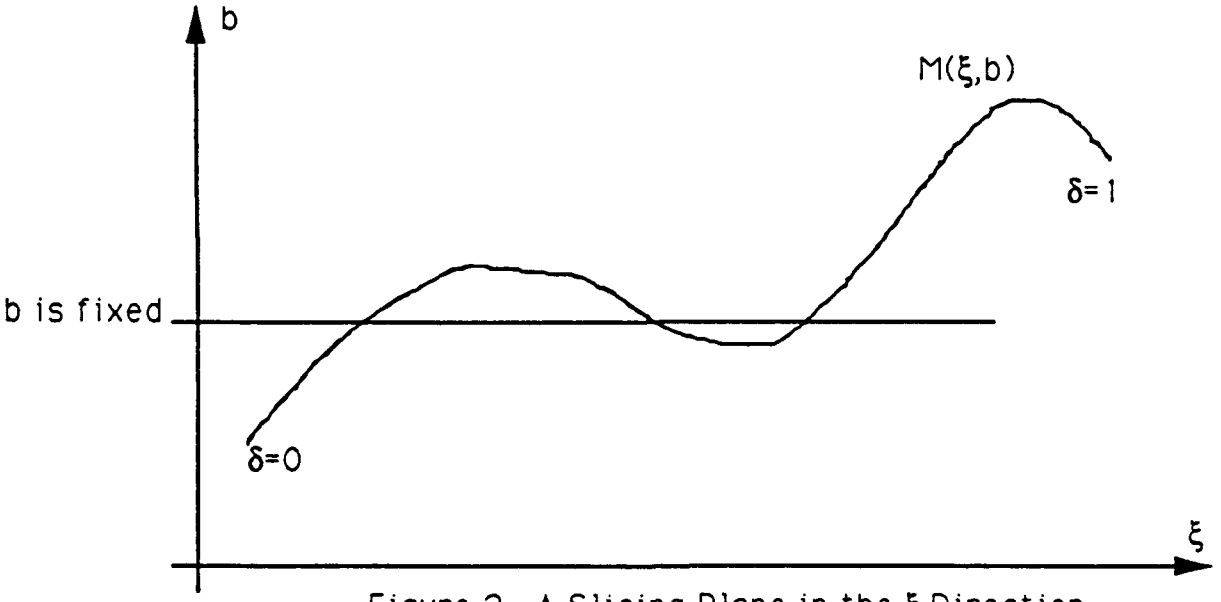


Figure 2. A Slicing Plane in the ξ Direction

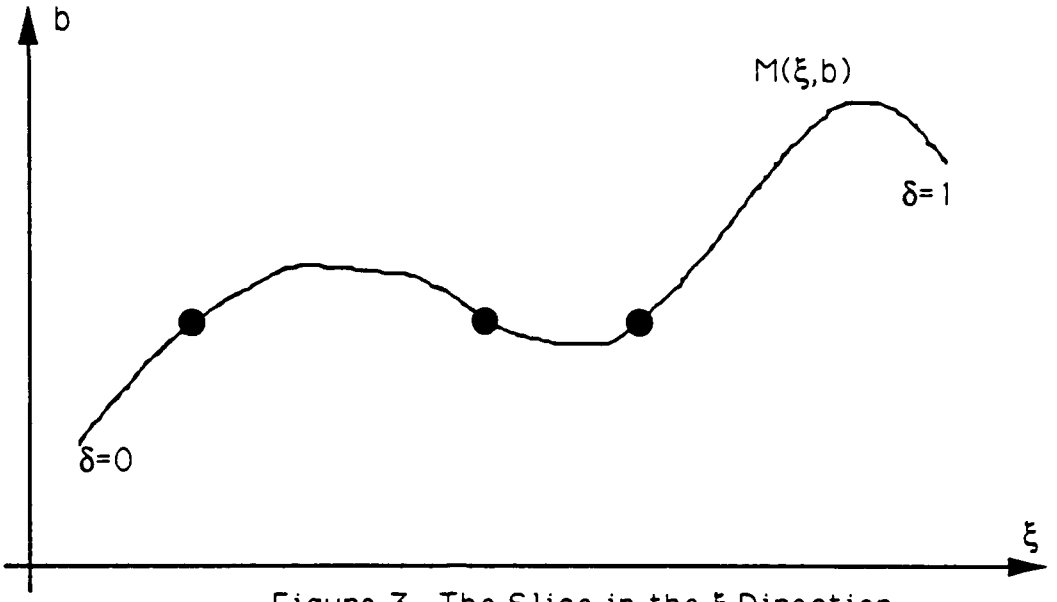


Figure 3. The Slice in the ξ Direction

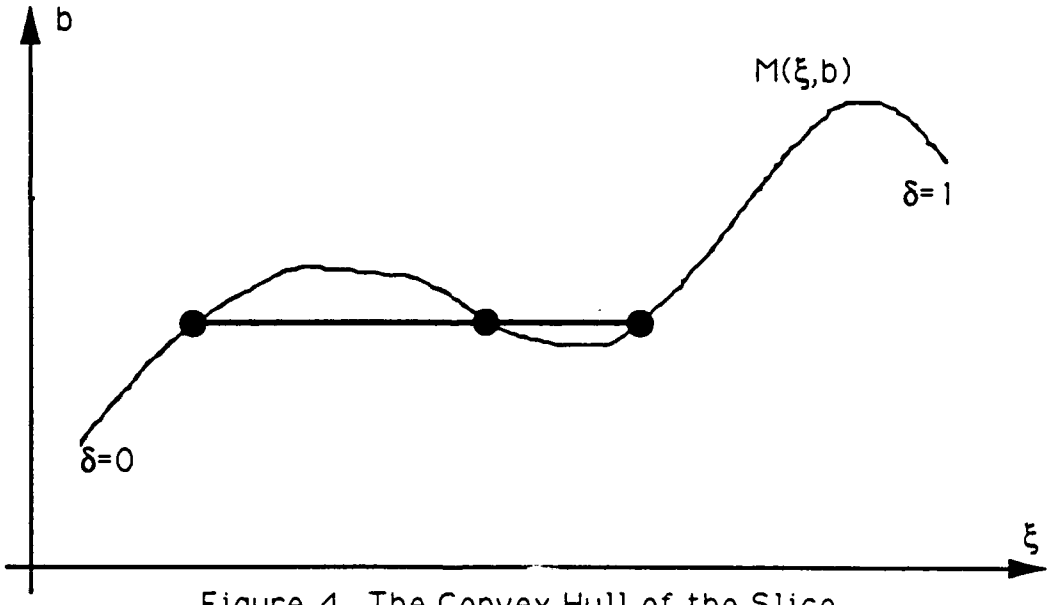


Figure 4. The Convex Hull of the Slice

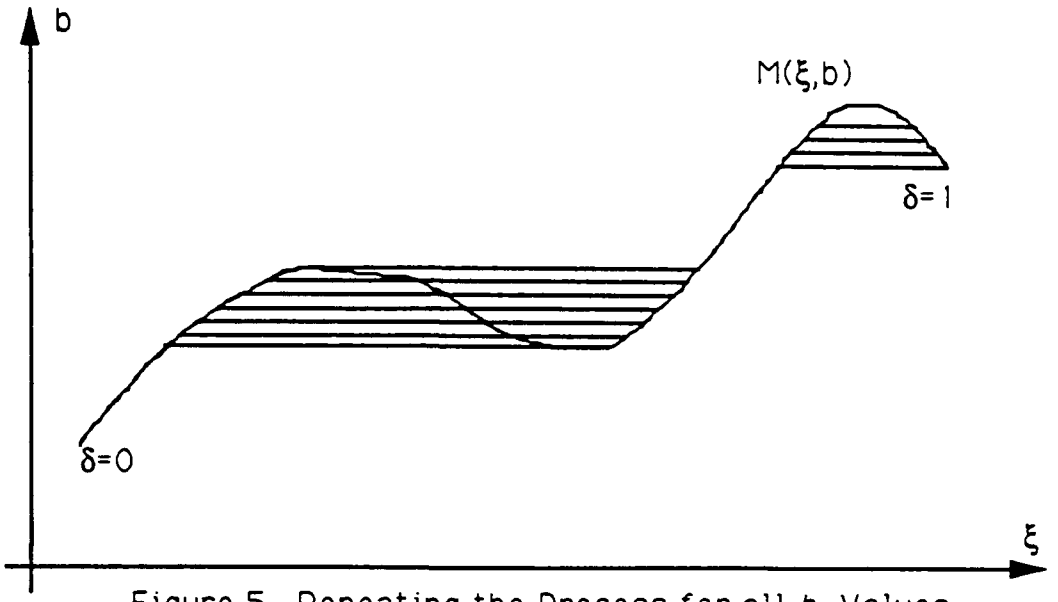


Figure 5. Repeating the Process for all *b* Values

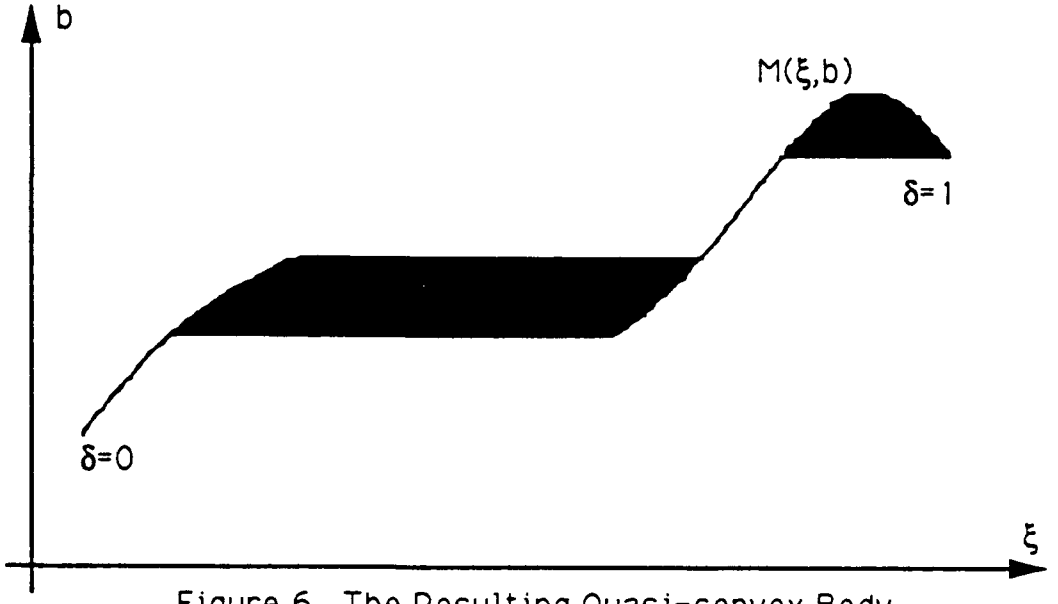


Figure 6. The Resulting Quasi-convex Body

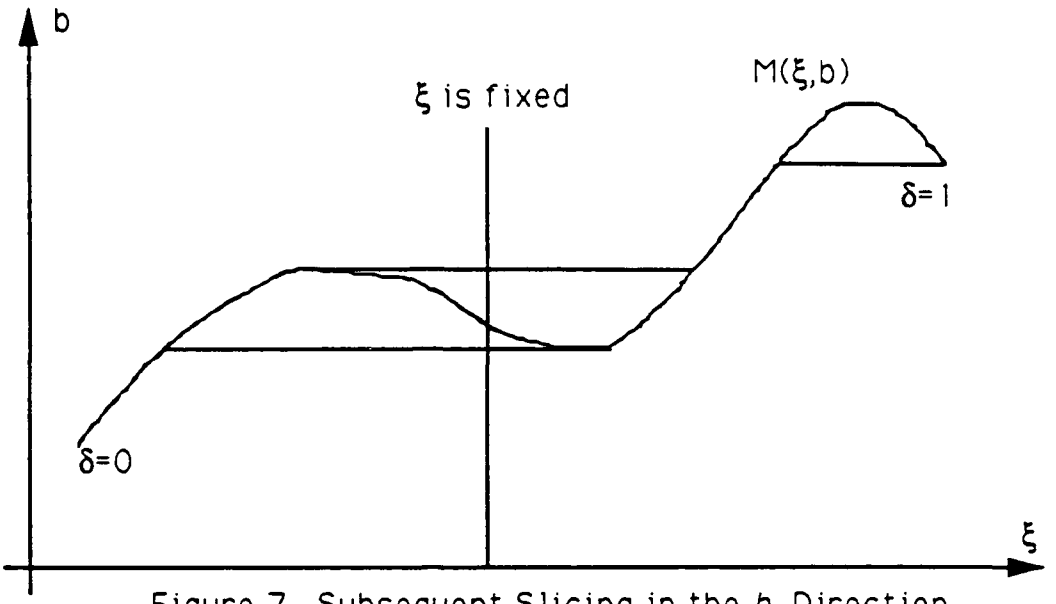


Figure 7. Subsequent Slicing in the b Direction

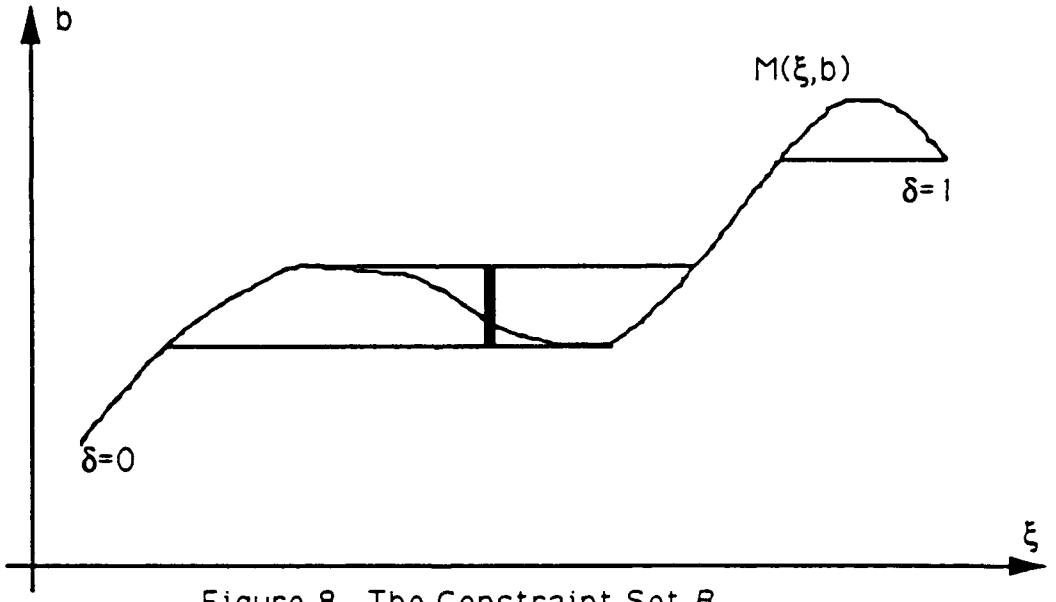


Figure 8. The Constraint Set B_ξ

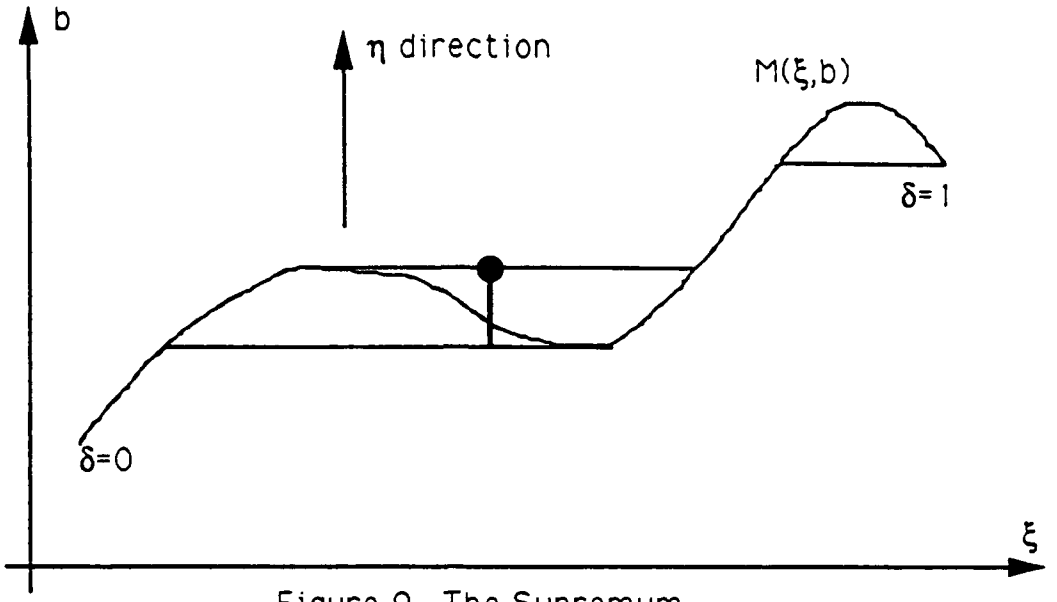


Figure 9. The Supremum

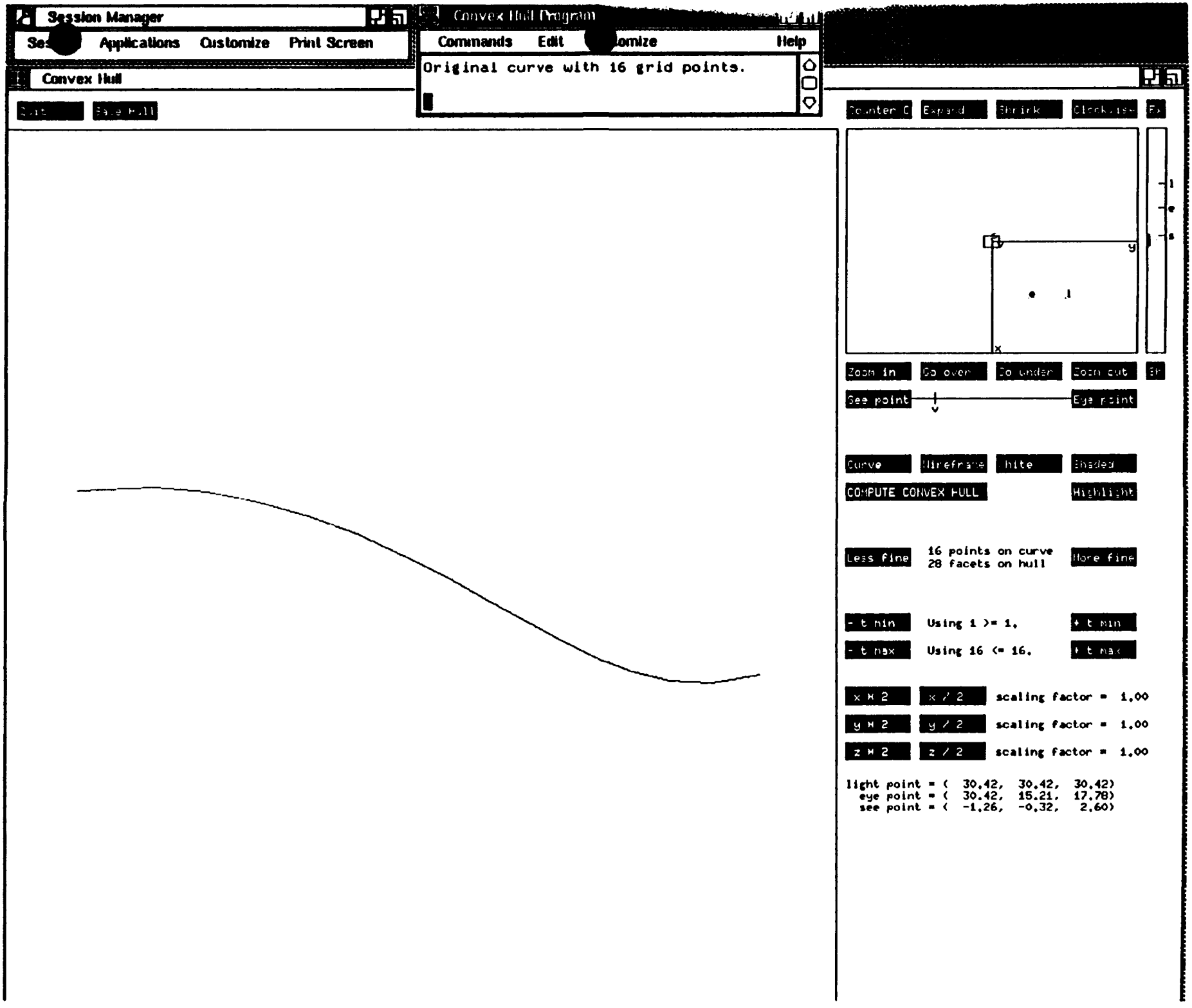


Figure 10. Original curve with 16 grid points

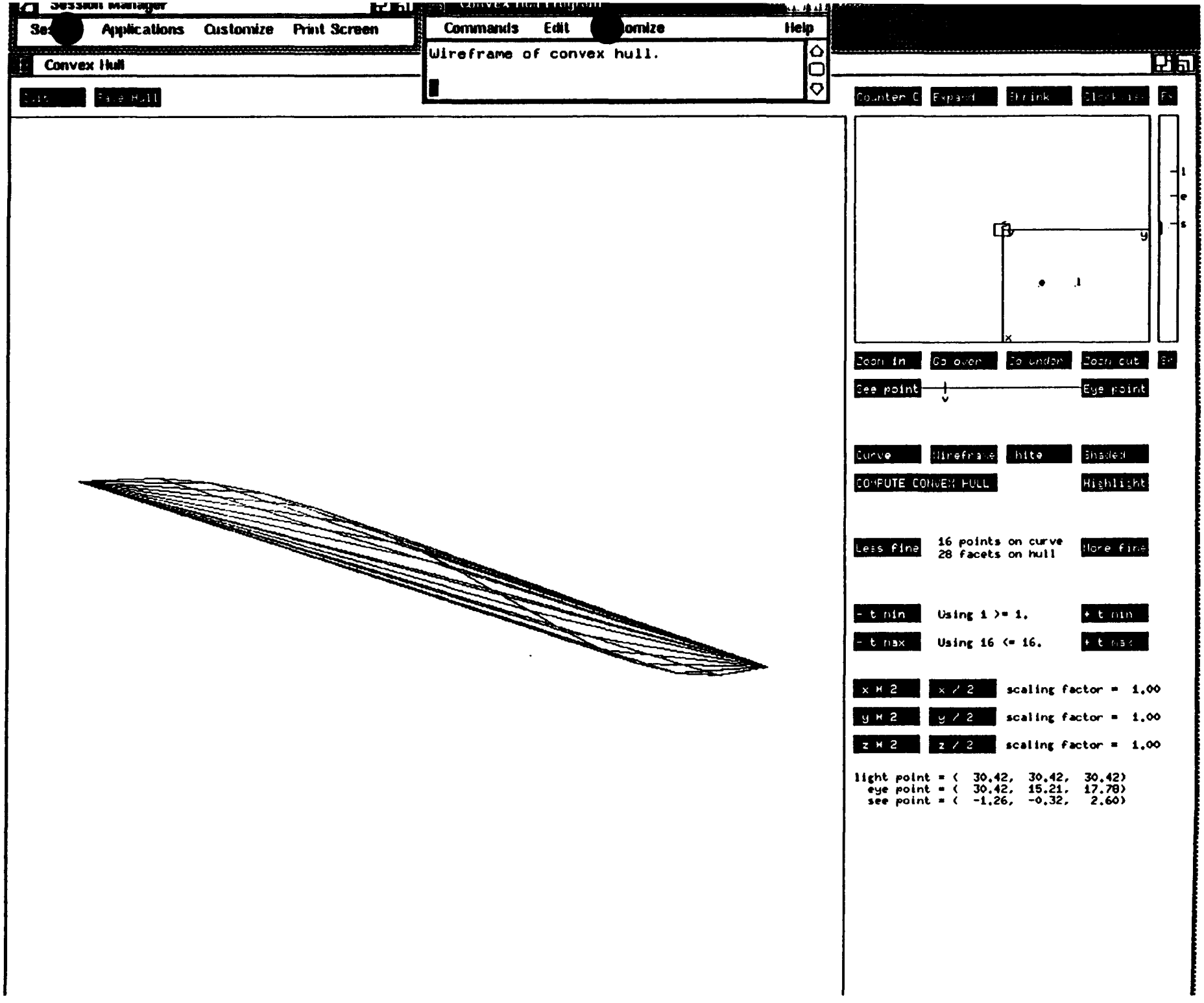


Figure 11. Wireframe of convex hull

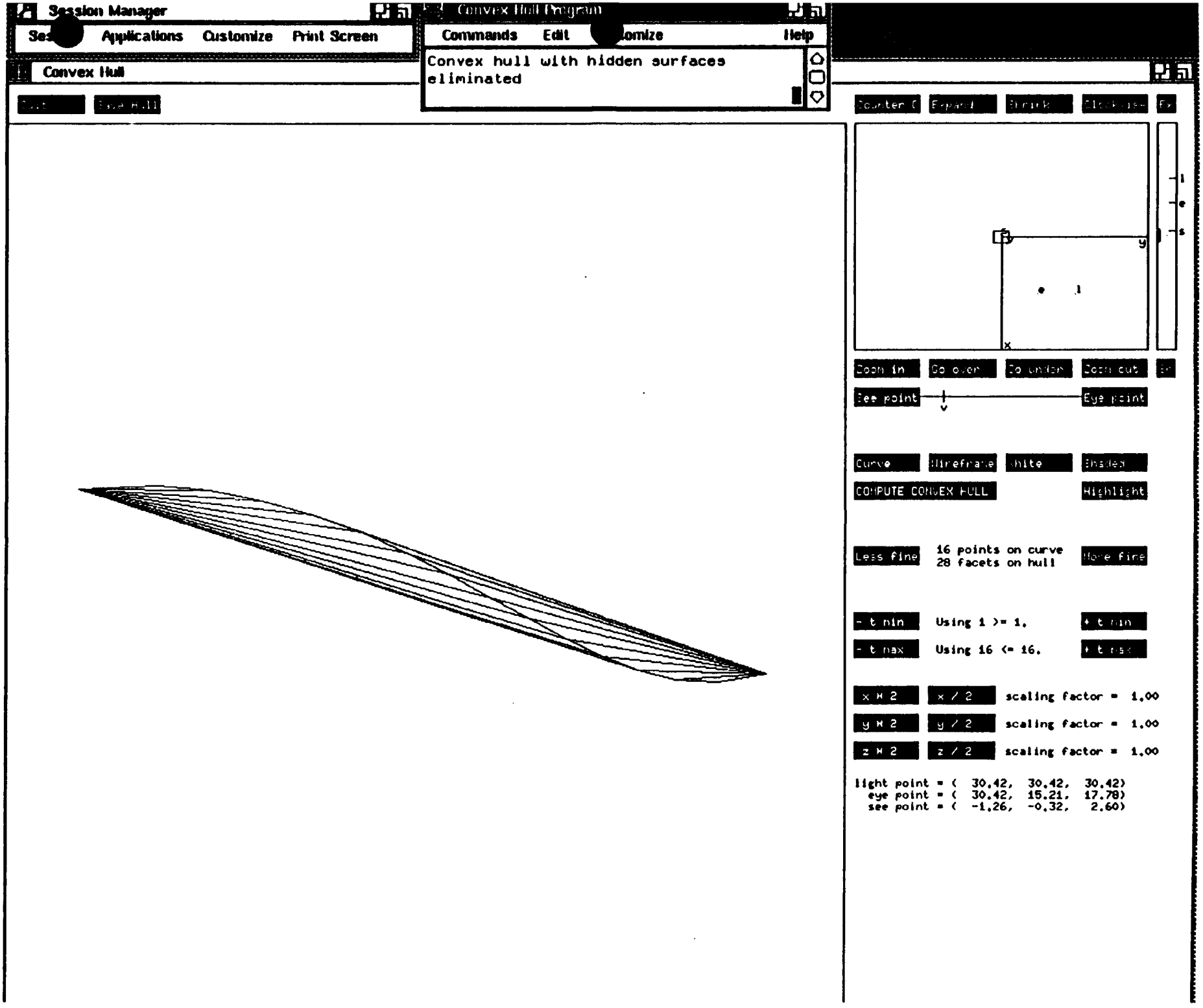


Figure 12. Convex hull with hidden surfaces eliminated

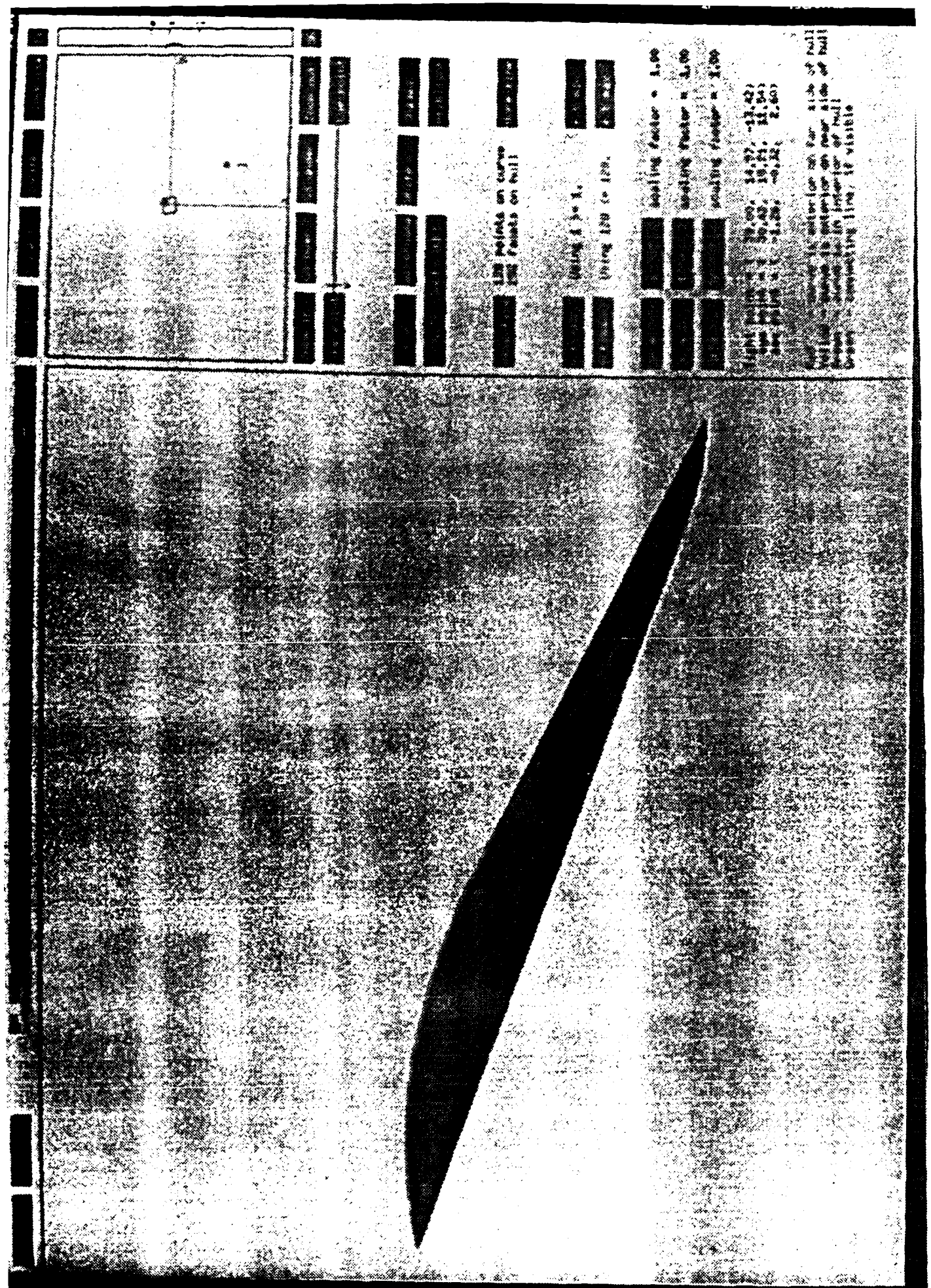


Figure 13. Shaded convex hull: 8 points on curve, 16 facets on hull

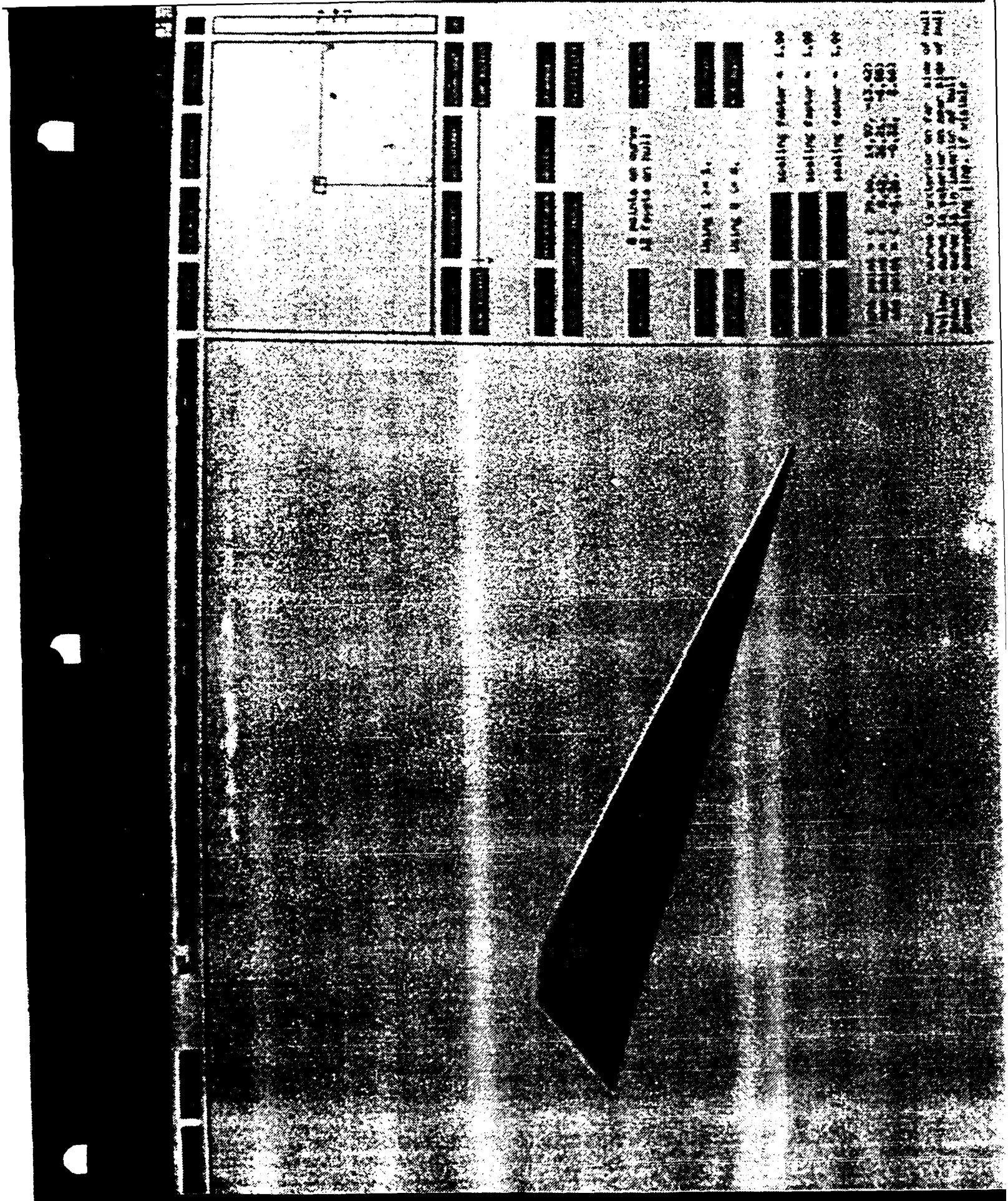


Figure 14. Shaded convex hull: 128 points on curve, 22 facets on hull

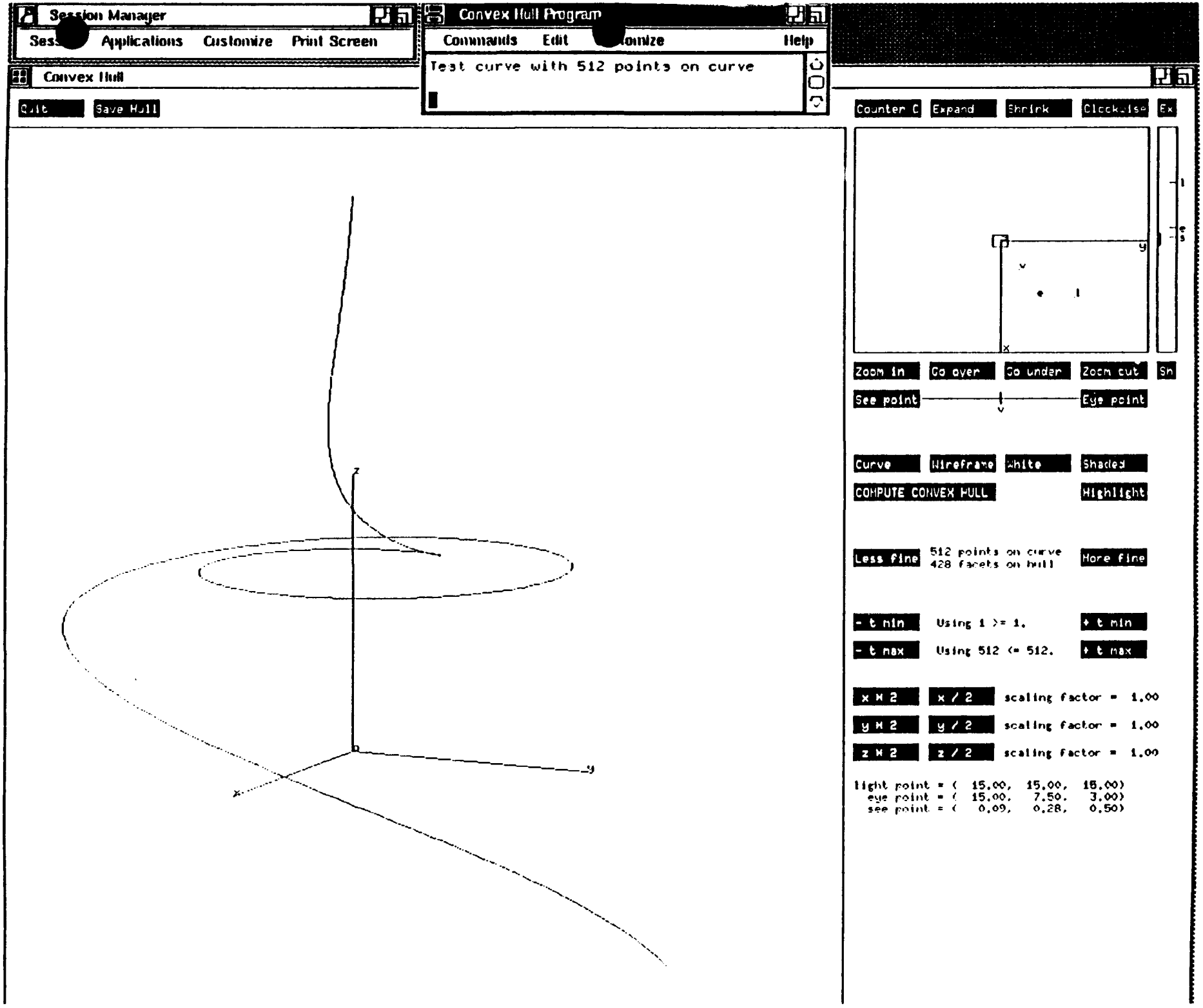


Figure 15. Test curve with 512 grid points

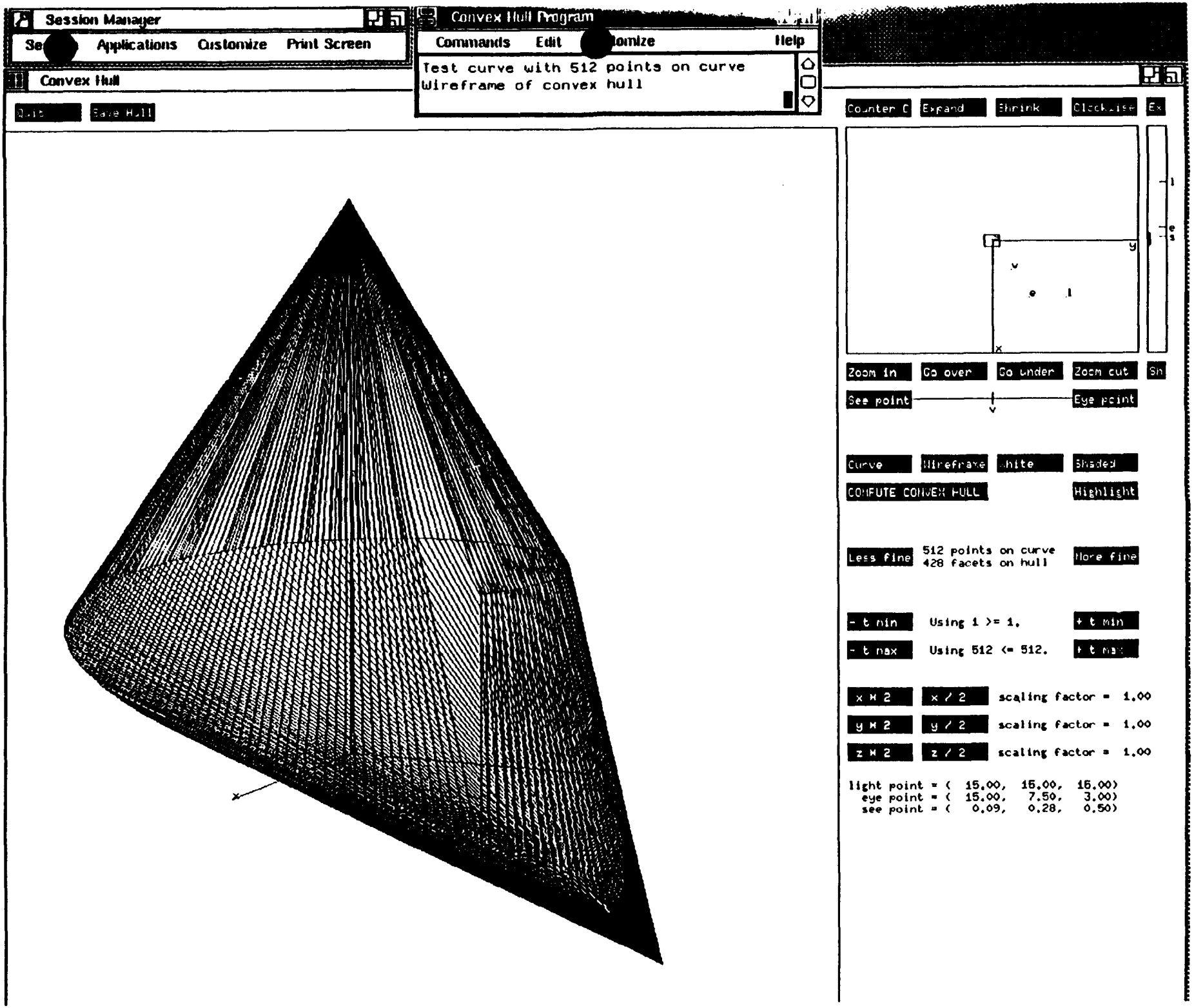


Figure 16. Wireframe of convex hull

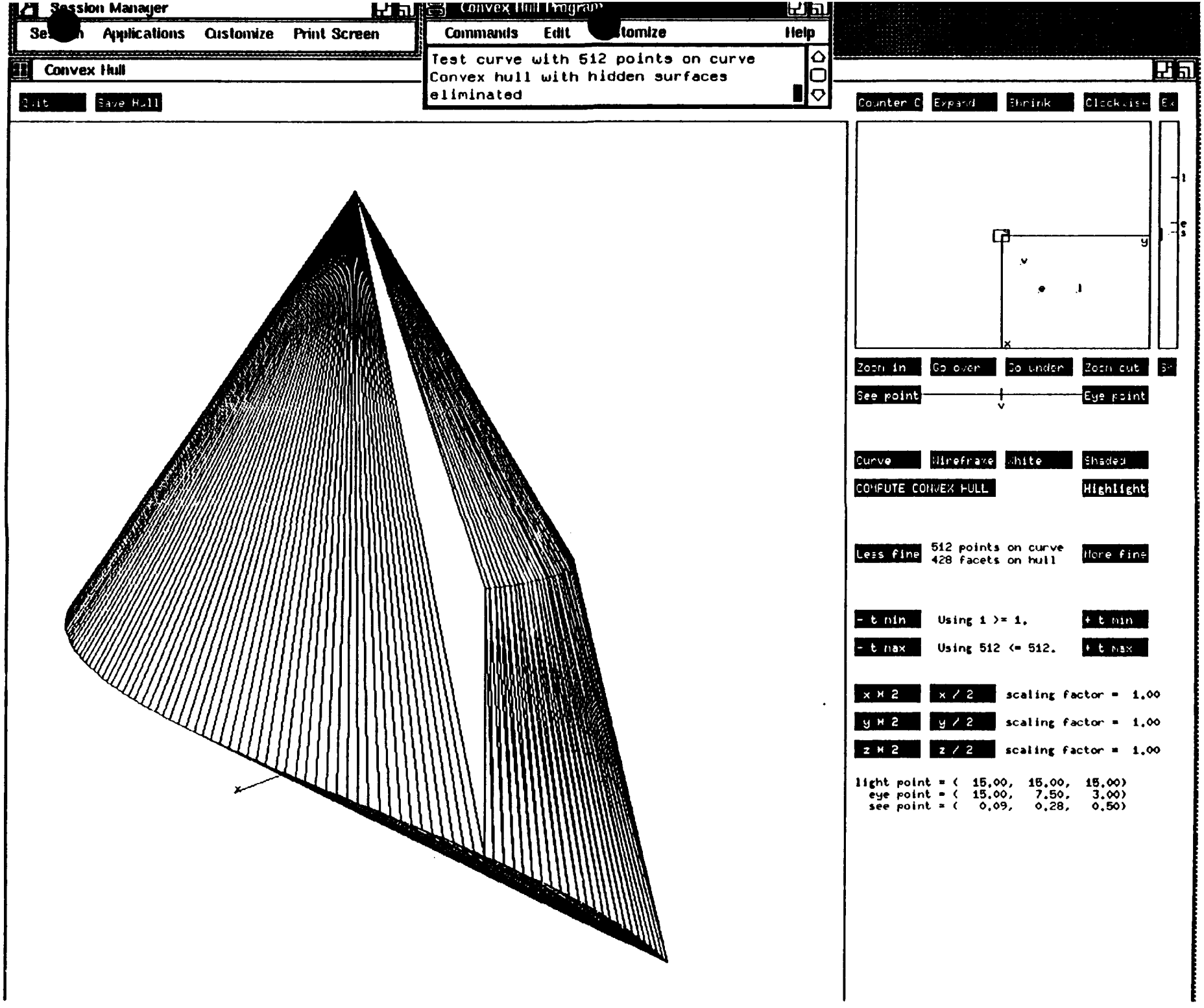


Figure 17. Convex hull with hidden surfaces eliminated

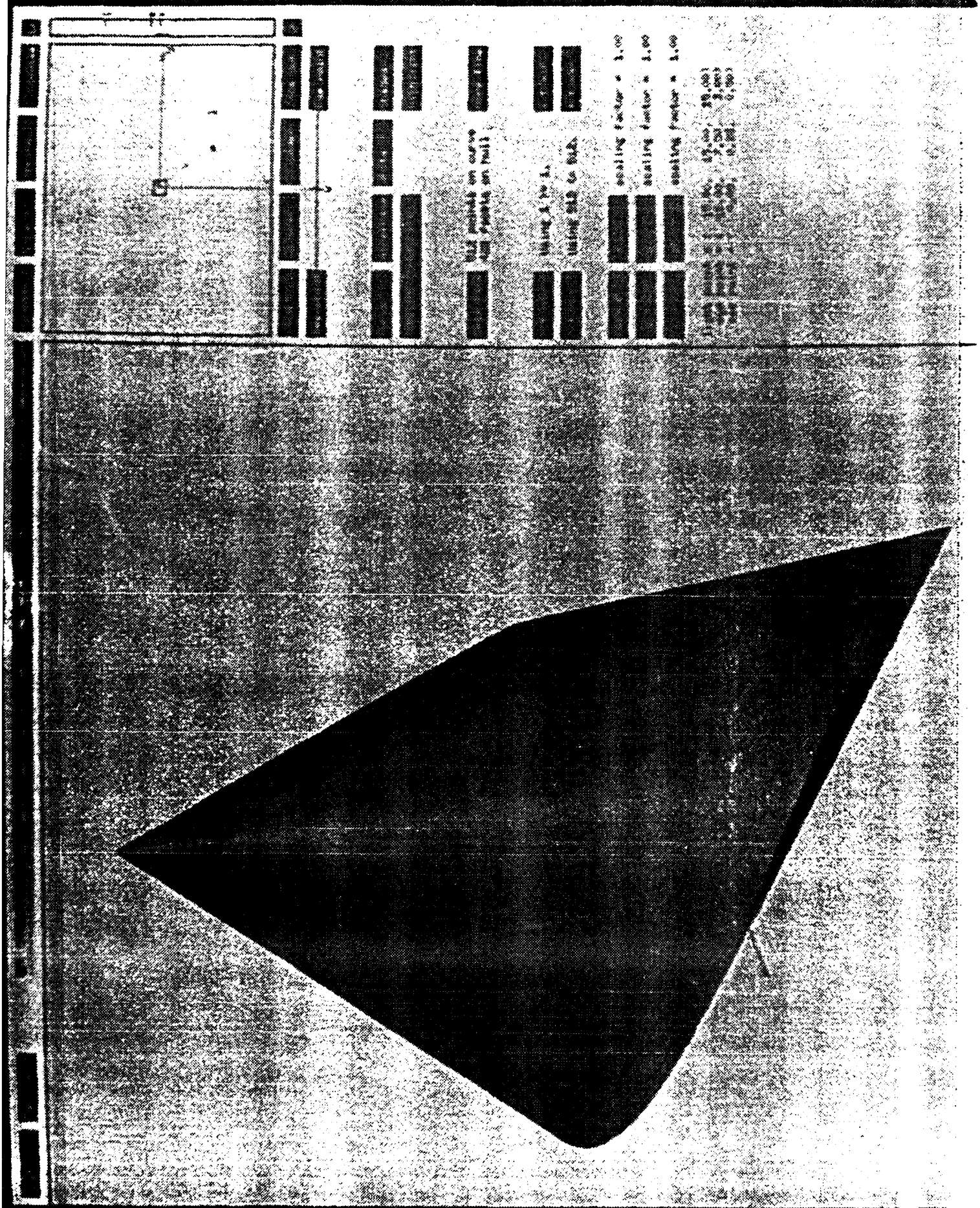


Figure 18. Shaded convex hull

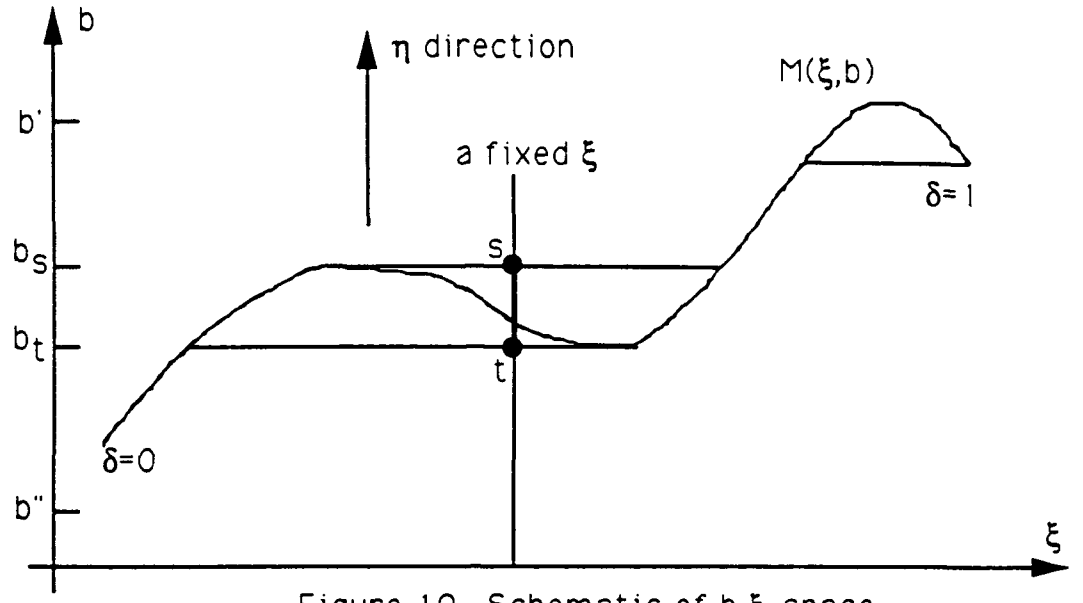


Figure 19. Schematic of b,ξ-space

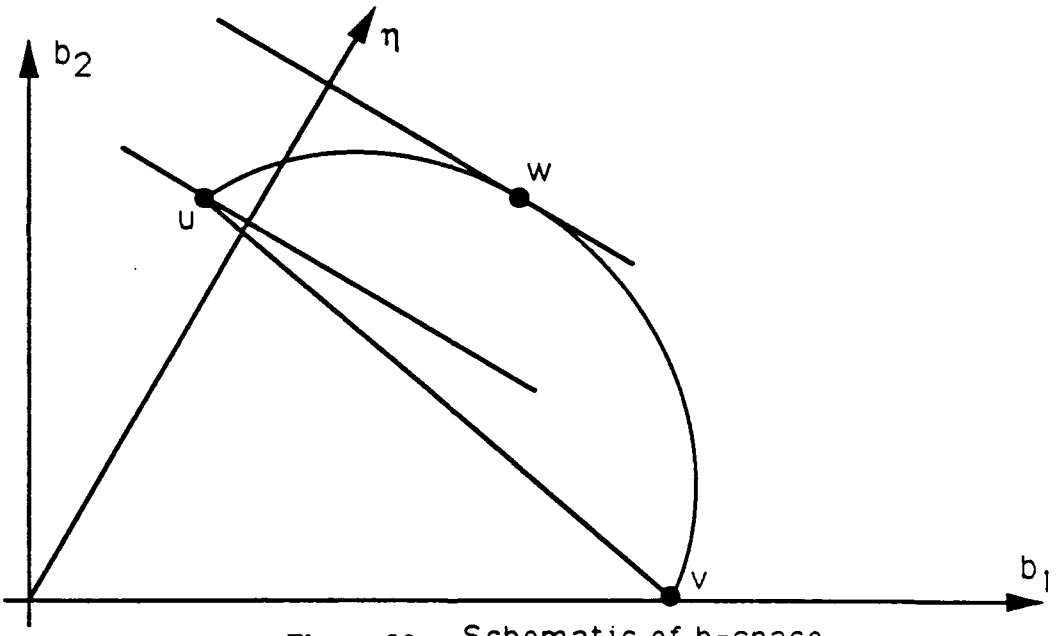
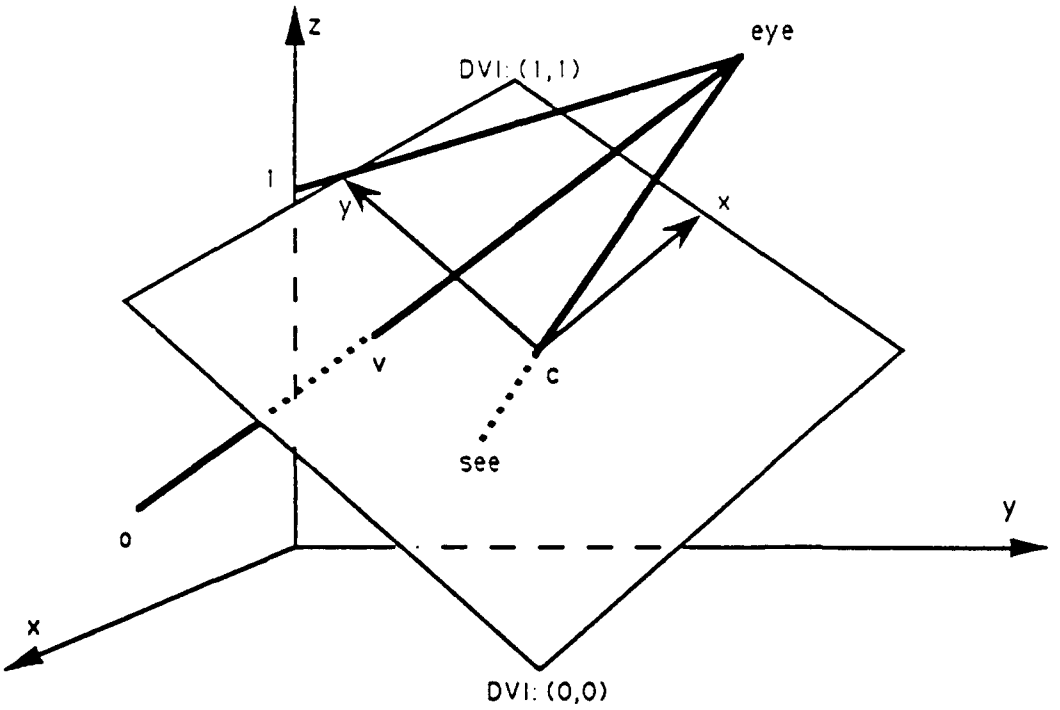


Figure 20. Schematic of b-space

Points *eye* and *see* are given as input.
The view plane is defined to be orthogonal to $\overline{eye,see}$.
The point *c* is defined to be the center of the view plane.
The view plane's y-axis is found by projecting the object space z axis back towards the *eye* point.
The view plane's x-axis is then perpendicular to both $\overline{eye,see}$ and its y-axis.



The point v can be represented in four different coordinate systems:
1) The three dimensional object space coordinates.
2) The two dimensional view plane coordinates.
3) The two dimensional device independent coordinates, with (0,0) at the lower left corner of the view rectangle and (1,1) at the upper right corner of the view rectangle.
4) The two dimensional device dependent coordinates, representing locations of pixels on the computer screen.

Figure 21. 3D to 2D Transformation

3 Some General Results Related to Equivalence Classes of Composites in 2D–elasticity and in the Theory of Plates

In the context of the reported effort, a further development was made of an observation [7] dated back to 1984 regarding the equivalence between strains arising in plates with tensors $\mathscr{D} + dT$, $d = const$ (see (36)) of stiffness.

A similar equivalence also holds for composite plates. Assume, for instance, that isotropic composite with moduli k,μ is generated by isotropic constituents with moduli k_1,μ_1 and k_2,μ_2, respectively. If we now apply the constituents with moduli $k_1 - d$, $\mu_1 + d$ and k_2-d, $\mu_2 + d$, $d = const$, then *the same composite* (i.e. composite of the same microgeometry) will possess moduli $k' = k - d$, $\mu' = \mu + d$. In other words, the Young's modulus $k' + \mu'$ of such composite will remain the same as before whereas the Poisson's ratio $\frac{\mu'-k'}{\mu'+k'} = \frac{\mu+d-(k-d)}{\mu+d+(k-d)} = \frac{\mu-k}{\mu+k} + \frac{2d}{\mu+k}$ will differ from the original value $\frac{\mu-k}{\mu+k}$. Particularly, this means that the effective Young's modulus for an isotropic elastic material containing voids is independent of the Poisson's ration of the matrix material [8]. This effect have been recently observed by Day, Snyder, Garboczi and Thorpe [9] as well as by Thorpe and Jasiuk [10], through numerical simulation. Various extensions of this result are about to come, specifically in the context of a shape optimization.

4 Status of the Research Effort

At this point the mathematical technique has been developed making it possible to analytically specify microstructures appearing in the optimal layout of materials for systems described by elliptic equations of the 2nd and 4th order with material constants treated as controls. The results obtained provide a theoretical basis for a subsequent implementation of a direct approach which promises drastic simplification in the numerical computation of optimal layouts. This computation will then be direct, i.e. based on the list of special laminar microstructures from which the global layout will be assembled with the aid of standard numerical procedures.

The effort in its present state has been shown to provide bounds and microstructures for several new situations never treated before. At the same time, the concept introduced here is expected to apply to a wide range of physical problems, including problems of optimal design. For this reason, a major theoretical development of this approach is anticipated. Specifically, stemming from the prior work, we expect to develop a general theory of quasisaddlification for integrands depending on two gradients. More precisely, the necessary and sufficient conditions for the integrand guaranteeing attainability of sup inf for functionals of the type (11) should be found. Secondly, the extension of the method to more than one physical field, i.e. fields of temperature and stress, etc., should be pursued. Also, the linkage between quasisaddlification and quasiconvexification should be investigated. All these issues will be treated in the sequel and will form the content of the renewal of this grant.

References

1. Lurie, K.A., "The Extension of Optimization Problems Containing Controls in the Coefficients", *Proc. Roy. Soc. Edinb.*, 114A, (1990), pp. 81–97.

2. Lurie, K.A., "On a General Concept in Optimal Material Layout", Lectures in Applied Mathematics, SIAM, (1991), pp. 301–321.

3. Lurie, K.A., and Lipton, R., "Direct Solutions of an Optimal Layout Problem for Isotropic Heat Conductors in Three Dimensions", in: *Theoretical Aspects of Industrial Design*, ed. by David A. Field and Vadim Komkov, SIAM, (1992), pp. 1–11.

4. Lurie, K.A., "Direct Solution of an Optimal Layout Problem for Isotropic and Anisotropic Heat Conductors on a Plane", *Journal of Optimization Theory and Applications*, Vol. 72, No. 3, (1992), pp. 553–575.

5. Lurie, K.A., "Direct Relaxation of Optimal Layout Problems for Plates", *Journal of Optimization Theory and Applications*, to appear.

6. Bendsoe, M., and Kikuchi, N., "Generating Optimal Topologies in Structural Design Using a Homogenization Method", *Computational Meth. Applied Mechanical Engineering*, Vol. 71, (1988), pp. 197–224.

7. Lurie, K.A., and Cherkaev, A.V., "A G–closure of Some Particular Sets of Admissible Material Characteristics for the Problem of Bending of Thin Elastic Plates", *Journal of Optimization Theory and Applications*, vol. 42, (1984), No. 2, pp. 305–316.

8. Cherkaev, A.V., Lurie, K.A., and Milton, G.W., "Invariant Properties of Stress in Plane Elasticity and Equivalence Classes of Composites", *Proc. Roy Soc. Lond.*, 438A, No. 1904 (1992), pp. 519–529.

9. Day, A.R., Snyder, K.A., Garboczi, E.J. and Thorpe, M.F., "The Elastic Moduli of a Sheet Containing Circular Holes", *J. Mech. Phys. Solids*, to appear.

10. Thorpe, M.F., and Jasiuk, I. "New Results in the Theory of Elasticity for Two–Dimensional Composites", *Proc. Roy. Soc. Lond.*, 438A, No. 1904 (1992), pp. 531–544.

Presentations

1. "Direct Solutions of an Optimal Layout Problem for Isotropic Heat Conductors in Three Dimensions", by K. Lurie and R. Lipton. Invited address presented at SIAM Regional Conference on Industrial Design Theory, AFIT, Wright–Patterson AFB, Ohio, April 1990.

2. "A Methodology of the Determination of Optimal Structure Characteristics of Elastic Bodies", by K. Lurie. Paper presented at Third Air Force/NASA Symposium on recent advanced in multidisciplinary analysis and optimization, San Francisco, California, September 1990.

3. "On a General Concept in Optimal Material Deployment", by K. Lurie. Presented at the seminar of Department of Mathematics, Brown University, October 1990.

4. "A New Method of Solution of Non–Self–Adjoint Problems of Optimal Design", by K. Lurie. Presented at the seminar of Courant Institute, December 1990.

5. "On a General Concept in Optimal Material Layout", by K. Lurie. Invited address presented at IUTAM Symposium on Optimal Design with Advanced Materials, Lyngby, Denmark, August 1992.

6. "Direct Relaxation of Optimal Layout Problems for Plates", by K. Lurie. Presented at the XVIIIth International Congress of Theoretical and Applied Mechanics, Haifa, Israel, August 1992.

List of Personnel Associated with the Research Effort

K. Lurie - Principal Investigator

J. Northrup - Co-Principal Investigator

Consultants:

Prof. Martin Bendsøe, Institute of Mathematics, Technical University of Denmark

Prof. Andrei Cherkaev, University of Utah

Prof. Leonid Gibianskii, Courant Institute

Prof. Robert Lipton, Worcester Polytechnic Institute

Prof. Graeme Milton, Courant Institute

Prof. Nina Ural'tseva, Emory University

List of Publications Associated with the Research Effort

Lurie, K.A., "On a General Concept in Optimal Material Layout", Lectures in Applied Mathematics, SIAM, (1991), pp. 301–321.

Lurie, K.A., and Lipton, R., "Direct Solutions of an Optimal Layout Problem for Isotropic Heat Conductors in Three Dimensions", in: *Theoretical Aspects of Industrial Design*, ed. by David A. Field and Vadim Komkov, SIAM, (1992), pp. 1–11.

Lurie, K.A., "Direct Solution of an Optimal Layout Problem for Isotropic and Anisotropic Heat Conductors in a Plane", *Journal of Optimization Theory and Applications*, Vol. 72, No. 3, (1992), pp. 553–575.

Lurie, K.A., "Direct Relaxation of Optimal Layout Problems for Plates", *Journal of Optimization Theory and Applications*, to appear.

Cherkaev, A.V., Lurie, K.A., and Milton, G.W., "Invariant Properties of Stress in Plane Elasticity and Equivalence Classes of Composites", *Proc. Roy. Soc. Lond*, 438A, No. 1904 (1992), pp. 519–529.

Lectures in Applied Mathematics
Volume 27 (1991)

On a General Concept in Optimal Material Layout

K. A. LURIE

Abstract. The proposed paper is intended to develop a methodology for the determination of optimal structural characteristics of elastic bodies designed for work in a variety of external conditions (load, static and dynamic regimes, loss of stability), or under the action of combined physical fields (stresses and temperature, electric and magnetic fields, etc.). By the term structural characteristics we mean the elastic constants, heat and electrical conductances and other similar parameters varying with position; also, the problems of optimal distribution of thickness of elastic constructions, i.e., plates and shells, and of holes and cavities in elastic bodies could as well be formulated along similar lines. The structural characteristics (controls) are in all cases assumed to admit values belonging to some admissible set U.

The cost (objective) functional is assumed to be any weakly semicontinuous functional of the solutions to the corresponding boundary value problem; also, it may depend explicitly on the design parameters. (The assumption of weak semicontinuity is rather nonrestrictive: it is satisfied for many typical cost functionals used in practice.) The constraints are imposed on the design parameters as well as on the variables characterizing the system's behavior relative to each physical field considered.

The method of solution is constructive; it provides an effective procedure of immediate transformation of optimization problems to their relaxed form, which ensures that they are well-posed, e.g., the existence of the optimal structural characteristics (controls). This fundamental step has hitherto been committed with the aid of the so-called G-closures of the original sets U of admissible controls. Since the G-closures are known for a very restricted set of examples and the construction of the new ones provides substantial difficulties, it is impractical to rely on them for problem relaxation. The technique developed in this study is intended to avoid any reference to G-closures at all.

1980 *Mathematics Subject Classification* (1985 *Revision*). Primary 49A, 73K.

0075-8485/91 $1.00 + $.25 per page

In essence, what is really needed is the specific relaxation adapted for the particular optimization problem considered. We propose below a constructive method of such a relaxation; based on that we also propose a numerical procedure, along with a computer implementation of the procedure, which will permit a designer to find the optimal materials layout and to present a required design in a form convenient for practical use. This new approach is more feasible because it requires significantly less information than the G-closure approach.

Introduction. The development of the general theory of structural optimization has by now passed through two stages. The first period may be called naive: it has been characterized by a firm belief in the power of the necessary conditions of optimality *per se* with almost no regard to the existence considerations. The necessary conditions were intended to describe potentially optimal regimes, and it only remained to assemble them to form an optimal pattern of materials. This reasoning has also motivated the wide-spread conviction that the preliminary discretization of the problem combined with subsequent use of nonlinear programming would generally make it possible to determine the optimal control.

This entire concept has turned out to be unjustifiable: first, it has been disproved by a thorough analysis of the necessary conditions [1] and later by a careful inspection of numerical procedures associated with the initial discretization [2], [3].

Physically speaking, failure of a naive approach is closely connected with the remarkable phenomenon of the appearance of *microstructures* in the process of formation of the optimal materials layout. This phenomenon may be illustrated by the following example related to the distribution of temperature in a heat-conducting medium.

Assume that we are given two isotropic materials with differing heat conductances; the materials should be placed in a given domain O so as to maximize a certain functional associated with the distribution of temperature caused by some fixed system of sources under some set of boundary conditions. For such a functional, one may choose the mean square difference between the actual and desired temperature distributions, or the heat flux across some particular part of the domain's boundary.

To obtain the required temperature distribution, it is necessary to facilitate the conditions for the heat to flow in some selected favorable direction, and to inhibit this flow along the perpendicular direction; all this must be done at every point in the region. This implies that the

heat conductance of the required medium must at every point be dependent on direction; i.e., it must be a tensor function of the coordinates, and the material itself must be anisotropic. The difficulty which now arises is that we have no appropriate anisotropic materials among the originally given compounds, which are themselves isotropic. The only alternative is to build the required medium *artificially* by assembling some microstructure from the given compounds. The simplest example is provided by a laminate composite; its effective conductances along and across the layers differ from one another. It must be added that the problem in question does not contain any parameter which might restrict thickness of initial materials from below, e.g., the width of layers in a laminate. We must therefore expect that the optimal value of a functional will be attained for an infinite partitioning of the domain (or some part of it) into parts occupied by various original compounds. (See Figure 1.)

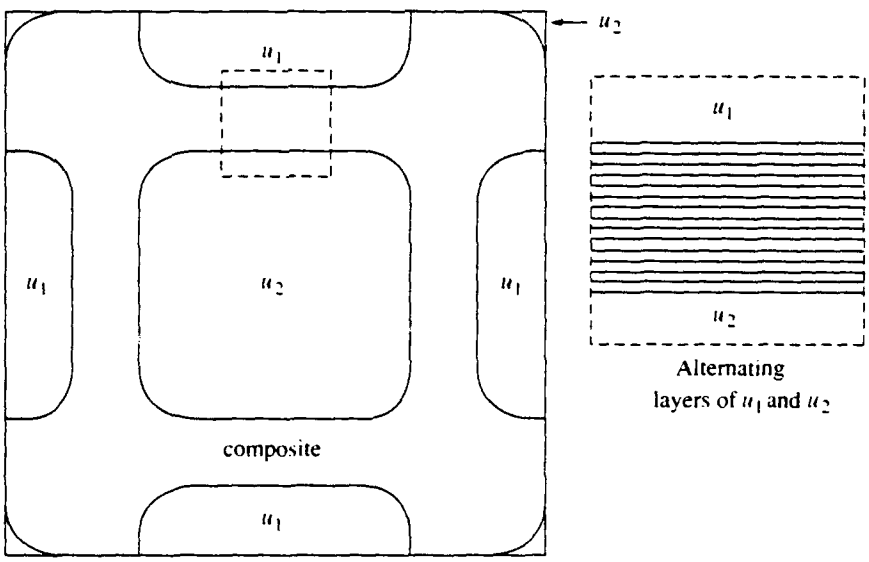


FIGURE 1. Optimal materials layout for torsion problem

Mathematically, the inconsistency of a naive statement could be made evident by a careful inspection of the set of necessary conditions of optimality. Consider for example the problem of torsional rigidity of an elastic prismatic rod of cross-section S; its torsion rigidity I is equal

to

$$I = 2\int_S w\,dx, \quad x = (x^1, x^2), \tag{1}$$

where w denotes Prandtl's function, i.e., the solution of the boundary value problem

$$\nabla \cdot D(x)\nabla w = -2, \quad w|_{\partial s} = 0, \tag{2}$$

$$D(x) = u(x)E. \tag{3}$$

Here, $u(x)$ denotes the elastic compliance of the rod's material at the point $x \in S$, and E denotes a unit tensor.

It is necessary to choose the function $u(x)$ so as to maximize the rigidity I of a rod if the mean value u_0 of its elastic compliance and the interval $[u_1, u_2]$ of the admissible values of $u(x)$ are fixed:

$$u_1 \le u(x) \le u_2, \tag{4}$$

$$\int_S u(x)\,dx = u_0\,\mathrm{meas}(S), \tag{5}$$

$$0 < u_1 < u_0 < u_2 < \infty. \tag{6}$$

This problem has been examined in [4] where it has been shown that the necessary condition of Weierstrass requires that the optimal layout can only include the limiting values u_1 and u_2 of compliance in accordance with the rule

$$u = u_1, \quad \text{if } (\nabla w)^2 \ge \kappa u_2/u_1, \tag{7}$$

$$u = u_2, \quad \text{if } (\nabla w)^2 \le \kappa u_1/u_2. \tag{8}$$

Here, the constant $\kappa > 0$ denotes the Lagrange multiplier associated with the integral constraint (5).

Because $u_1/u_2 < 1$, from (7), (8) we deduce that none of the stationary regimes can be optimal provided that the values $(\nabla w)^2$ belong to the banned interval $(\kappa u_1/u_2, \kappa u_2/u_1)$. On the other hand, those parts of S which are occupied by u_1 and u_2 materials are separated by some line Γ with normal $\mathbf{n}$ and tangent $\mathbf{t}$; across this line, the value of $(\nabla w)^2$ suffers a jump. The latter can be determined from the continuity conditions

$$[\nabla w \cdot \mathbf{t}]_1^2 = 0, \tag{9}$$

$$[u\nabla w \cdot \mathbf{n}]_1^2 = 0, \tag{10}$$

where $[\cdot]_1^2 = [\cdot]_2 - [\cdot]_1$ denotes the jump of a quantity within the square brackets.

From inequality (7) taken at some point close to Γ on that side of Γ where $u = u_1$ we deduce (bearing (9), (10) in mind) that

$$\kappa \leq \frac{u_1}{u_2}[(\nabla w)^2]_1 = \frac{u_2}{u_1}[(\nabla w)^2]_2 + [\nabla w \cdot \mathbf{t}]^2 \left(\frac{u_1}{u_2} - \frac{u_2}{u_1}\right). \tag{11}$$

The latter condition can be made compatible with inequality (8) only provided that

$$\nabla w \cdot \mathbf{t} = 0. \tag{12}$$

Both inequalities (7), (8) are then fulfilled as strict equalities on each side of Γ.

The latter curve should thus satisfy both conditions (7) and (8) simultaneously; this makes the problem of finding it overdetermined and therefore contradictory. Formally, the situation is as if the position of this curve and its slope were to be determined each from a separate independent equality. *Such a problem is known to be unsolvable in a class of smooth curves.* One may expect that the solution might exist among the generalized curves whose windings would be dense within a set of nonzero measure. The correct layout is in fact illustrated in Figure 1.

The latter observation has found support in the analysis of numerical data associated with the attempts to apply nonlinear programming to the originally discretized version of the optimization problem. The numerical procedure has failed to display any evidence of convergence; rather, it has demonstrated fast oscillating behavior of the materials' layout. Increasing the accuracy of the calculations and refining the discretization may lead to a substantial instability resulting in a completely different pattern. These observations, [2], [3], have shown that the computational procedure should be chosen in accordance with existence considerations which could be the only ones to guarantee necessary convergence of a numerical scheme.

In the course of successive approximations to the optimal pattern, the so-called *chattering* regimes of control have been discovered to appear. Applied to problems like those described above (and many related ones containing controls in coefficients, see equation (2)) this implies the appearance of infinitely many small zones occupied by different regimes of control (i.e., by different materials). The interfaces of such subregions form a set which is dense within some well-expanded part of the original domain. In this specific sense, we could speak about generalized curves separating the two types of material. This is nothing but a realization of an old idea by L. C. Young, [5], applied to the specific type of

problems considered here. Such a layout is nothing but what has been identified above as a microstructure; we have thus justified numerically the physically expected distribution of controls.

The chattering regimes appear almost inevitably whenever we attempt to build up the optimal layout of two or more materials. This is because the multidimensional problems, e.g., problems related to rods, plates, shells, and three-dimensional bodies, are associated with vector and tensor fields (currents, strain, stress, etc.), and the corresponding optimization problems are concerned with the optimal formation of such fields. The latter are associated with one or more advantageous directions at each point; for this reason, to provide an effective control, we need a certain type of anisotropic medium, and this is generally not at the designer's disposal. To form it up, one has to introduce chattering regimes, and this is what really happens when we construct a minimizing sequence of controls.

In other words, the chattering regimes appear whenever the original set of admissible controls does not contain the required anisotropic medium; mathematically, we say that this set does not possess some specific type of closure property. Applied to the optimization problems considered here, it means that the set of admissible controls should include, along with the original constituents, all the composites assembled from them. This new set of controls, emerging from the original set U, is called the G-closure of U and designated GU (see [7] where this notion was first introduced). The set U is called G-closed if $U = GU$; generally, U is contained in GU. In many applications, the original sets U of controls are not G-closed, and the problem arises of constructing their G-closures. For a number of important examples, mostly associated with the second-order operator $\nabla \cdot u\nabla$, G-closures have been built explicitly, [8]–[19]. Associated with this problem is the second stage of the theory's development. The reason is that the optimal control generally belongs to GU rather than to U. In other words, the necessary conditions of optimality, among them the Weierstrass condition, would now be noncontradictory provided that the admissible set U of controls coincides with its G-closure.

This approach will be illustrated by an example from the theory of heat. Consider the following problem of optimal control [6].

We consider a plane rectangular domain $(-a \le x \le a, 0 \le y \le 1)$ (see Figure 2; note that line segments dividing the regions in this figure might in fact be curved arcs). Across its upper boundary Γ_1 $(y = 1)$ there flows a uniform flux of unit intensity, and other parts of the bound-

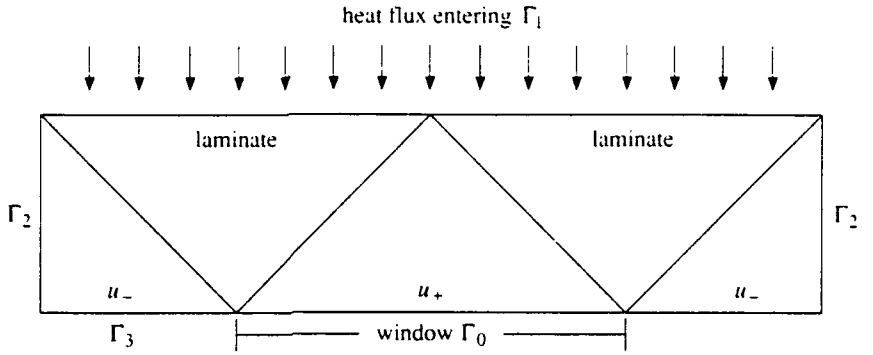


FIGURE 2. Heat flux being focused towards a window

ary are either thermoisolated, i.e., free of heat flux (these parts will be designated by Γ_2), or the temperature along them will be kept equal to zero (such parts will be denoted by Γ_3). The temperature distribution is assumed to depend only on the coordinates x, y. The domain is occupied by two isotropic materials whose specific thermal conductivities are given by u_- and u_+ respectively, with $0 < u_- < u_+ < \infty$. The temperature distribution can be found from the boundary value problem

$$\mathbf{q} = -D(x, y) \cdot \nabla T, \tag{13}$$

$$\nabla \cdot \mathbf{q} = 0, \tag{14}$$

$$D(x, y) = [u_- \chi_-(x, y) + u_+ \chi_+(x, y)]E, \tag{15}$$

$$\mathbf{q} \cdot \mathbf{n} = -1 \quad \text{on } \Gamma_1, \tag{16}$$

$$\mathbf{q} \cdot \mathbf{n} = 0 \quad \text{on } \Gamma_2, \tag{17}$$

$$T = 0 \quad \text{on } \Gamma_3, \tag{18}$$

where $\mathbf{n}$ denotes a unit vector along the outer normal to the boundary, E a unit tensor and χ_-, χ_+ characteristic functions of the subdomains S_- and S_+ of S occupied by the u_- and u_+ materials, respectively,

$$\chi_+^-(x, y) = \begin{cases} 1 & \text{if } (x, y) \in S_+^-, \\ 0 & \text{if } (x, y) \notin S_+^-. \end{cases} \tag{19}$$

We desire to distribute the materials in such a way that the functional

$$I = \int_{\Gamma_3} \rho(\Gamma) \mathbf{q} \cdot \mathbf{n}(\Gamma)\, d\Gamma \tag{20}$$

attains its maximum value. Here, $\rho(\Gamma)$ denotes a weight function, and

might be chosen, for instance, so as to "focus" the heat flux onto some portion of Γ_3.

Particularly, if this function is given by

$$\rho(\Gamma) = \begin{cases} 1 & \text{if } (x, y) \in \Gamma_0 \subset \Gamma_3 \\ 0 & \text{if } (x, y) \notin \Gamma_0 \end{cases} \tag{21}$$

then the problem reduces to that of maximization of a heat flux through the "window" Γ_0 on the boundary of a plane domain. (See Figure 2.)

To determine the optimal distribution of materials, we will introduce the G-closure of a set U of controls; the latter is defined here as

$$U = \{u_-, u_+\}. \tag{22}$$

The GU-set is the set of tensors $D_0 = d_1 e_1 e_1 + d_2 e_2 e_2$ of effective heat conductances of all composites assembled from the elements of U. The invariant description of GU is given below by the following inequalities [12], [14] (see hatched region in Figure 3):

$$u_- \le d_1 \le \frac{u_- u_+}{u_- + u_+ - d_2} \le d_2 \le u_+. \tag{23}$$

Particularly, for $d_1 = u_- u_+/(u_- + u_+ - d_2)$ we have laminates as elements of GU.

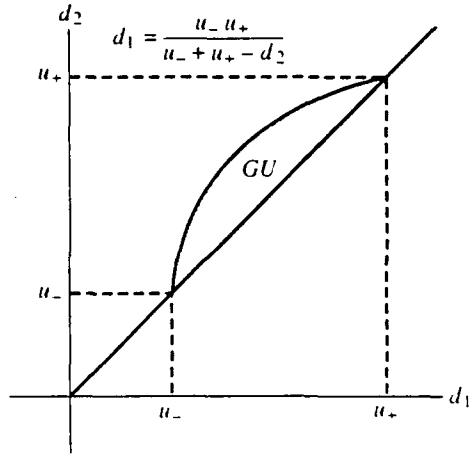


FIGURE 3. Invariant description of GU

In order to maximize the value of I we should choose among the elements of GU the corresponding composites to be placed at each point of the domain. In other words, we have to find the proper point (λ_1, λ_2)

within the figure in the (d_1, d_2)-plane restricted by a set of inequalities (23), as well as the angle φ which characterizes the orientation of the unit vectors $\mathbf{e}_1$ and $\mathbf{e}_2$:

$$\mathbf{e}_1 = \mathbf{i}\cos\varphi + \mathbf{j}\sin\varphi, \tag{24}$$

$$\mathbf{e}_2 = -\mathbf{i}\sin\varphi + \mathbf{j}\cos\varphi. \tag{25}$$

The original problem (13)–(21) should now be reformulated; instead of equations (13), (14) we must use the relationships

$$\mathbf{q} = -D_0(x, y)\cdot\nabla T, \quad \nabla\cdot\mathbf{q} = 0, \tag{26}$$

with $D_0(x, y)$ subject to the inequalities (23).

Making use of the conventional procedure of the calculus of variations we now introduce the augmented functional

$$J = \int_{\Gamma_3} \rho(\Gamma)\mathbf{q}\cdot\mathbf{n}(\Gamma)\,d\Gamma + \iint_S \lambda\nabla\cdot D_0\cdot\nabla T\,dx\,dy \tag{27}$$

where $\lambda = \lambda(x, y)$ is the Lagrange multiplier, taking into account the heat equation

$$\nabla\cdot D_0\cdot\nabla T = 0. \tag{28}$$

The conjugate system for λ is written down in the form

$$\nabla\cdot D_0\cdot\nabla\lambda = 0, \tag{29}$$

$$\mathbf{n}\cdot D_0\cdot\nabla\lambda = 0 \quad \text{along } \Gamma_1 \text{ and } \Gamma_2, \tag{30}$$

$$\lambda = \rho(\Gamma) \quad \text{along } \Gamma_3. \tag{31}$$

The necessary conditions show that the second inequality (23) must in fact be an equality; in other words, stationary composites may only be laminates. If we denote by 2χ the angle between the vectors ∇T and $\nabla\lambda$ then the necessary conditions dictate the following classification of regimes:

$$\lambda_1 = \lambda_2 = u_+ \quad \text{if } \tan\chi \le \sqrt{u_-/u_+}, \tag{32}$$

$$\lambda_1 = \lambda_2 = u_- \quad \text{if } \tan\chi \ge \sqrt{u_+/u_-}, \tag{33}$$

$$u_+u_-/d_1^2 = \tan^2\chi \quad \text{if } \sqrt{u_-/u_+} < \tan\chi < \sqrt{u_+/u_-}. \tag{34}$$

The optimal distribution of materials is thus characterized by a zone of highly conducting u_+ material provided that the directions of the gradients $\nabla\lambda$ and ∇T are close to each other, by a zone of low conducting u_- material if these directions are almost antiparallel, and by a zone of anisotropic laminate if the vectors are almost perpendicular

to each other (see Figure 2). Within this latter zone, the layers bisect the angle between the two gradient vectors; the optimal laminate tends to rotate the vector of heat flux to maximum possible extent.

One can observe that for a relaxed statement we obtain *three* regimes of control instead of two, as in a naive formulation of the problem. The new regime (34) is associated with a composite built from the original components. The domain filled in by a composite corresponds to the interval $(\sqrt{u_-/u_+}, \sqrt{u_+/u_-})$ of $\tan\chi$ which was prohibited in the nonrelaxed formulation.

This example illustrates how the optimal control could be evaluated provided that the GU-set has already been constructed starting from a given U-set.

Such a construction of GU is itself a difficult problem: it has been solved only for a few examples (almost all of which are listed in [6]). Describing G-closure is the same as describing a body in the space of invariants of the tensor of effective constants characterizing all possible composites assembled from the original elements in the U-set. The space of invariants can itself be high-dimensional; its dimensionality equals 18 for a general tensor of elastic constraints in three-dimensional problems. The body in question should have as its boundary *a manifold in* 17 *dimensions*. Above that, this manifold will be piecewise continuous, since various parts of it are described in different analytical terms. In summary, one will arrive at the conclusion that the use of G-closures for obtaining analytical information about optimal regimes is far from being practical.

At the same time, for many applications there is no need to know the G-closure in full. If an elastic body is subjected to some fixed system of loads, then with a corresponding optimal design we shall associate some well-defined field e of strain. The tensor D of elastic constants enters the problem only through Hooke's law $\sigma = D \cdot e$, e.g., through its projection along the strain e. We therefore need not know the entire tensor D; rather we require some linear combination of its components. For this reason one could look for a method of relaxation which would provide us with exactly the required combination, *without any reference to G-closure* since this set is not necessary for our specific purposes. We will see that this will be associated with a substantial reduction in dimensionality and therefore will leave more computational resources for considering more sophisticated problems, e.g., those involving a variety of external conditions.

New approach [20]. This approach will be illustrated by the same example as used to explain the G-closure approach. This time, however, we will not refer to the G-closure given in this specific case by inequalities (23).

We start with a reformulation of the constrained optimization problem (13)–(21) in terms of a max-min control problem. Introducing the Lagrange multiplier $\lambda(x, y)$ corresponding to the equation $\nabla \cdot D \cdot \nabla T = 0$ it is easy to show that the problem

$$\sup_{u} I \tag{35}$$

under the additional constraints (13)–(18) is equivalent to

$$\sup_{u,T} \inf_{\lambda} J \tag{36}$$

under the side conditions (18) and

$$\lambda|_{y=0} = \rho(x). \tag{37}$$

Here, the functional J is defined as

$$J = -\int_{-1}^{1} \lambda(x, 1)\, dx + \int_{s} u \nabla\lambda \cdot \nabla T\, dx\, dy. \tag{38}$$

For the functional (38) we will construct two types of estimates. The upper estimate will be built with the aid of a special mathematical technique, i.e., the combination of a preliminary estimate of the type $\sup \inf J \leq \inf \sup J$, followed by an additional estimate based on a new transform of the integrand. This transform provides a new function which is pointwise greater than or equal to the original. We have specifically

$$\begin{aligned} \sup_{u,T} \inf_{\lambda} J &= \sup_{T} \sup_{u} \inf_{\lambda} J \leq \sup_{T} \inf_{\lambda} \sup_{u} J \\ &= \sup_{T} \inf_{\lambda} \left[-\int_{-1}^{1} \lambda(x, 1)\, dx + \int_{s} G(\nabla T, \nabla\lambda)\, dx\, dy \right], \end{aligned} \tag{39}$$

$$G(\zeta, \eta) \triangleq \begin{cases} u_{+}\xi \cdot \eta & \text{if } \xi \cdot \eta \geq 0 \\ u_{-}\xi \cdot \eta & \text{if } \xi \cdot \eta \leq 0; \end{cases} \tag{40}$$

here we introduced the notation

$$\xi = \nabla T, \qquad \eta = \nabla\lambda. \tag{41}$$

Inequality (39) will be strengthened if we apply the transform

$$G^{**}(\xi,\eta) \triangleq \sup_A \sup_b \inf_a \{a\cdot\xi + b\cdot\eta + A(\xi_1\eta_2 - \xi_2\eta_1) - \inf_\xi \sup_\eta [a\cdot\xi + b\cdot\eta + A(\xi_1\eta_2 - \xi_2\eta_1) - G(\xi,\eta)]\} \tag{42}$$

and use the property

$$G^{**}(\xi,\eta) \geq G(\xi,\eta) \tag{43}$$

which is valid provided that $G(\xi,\eta)$ is convex in the η-variable (which is the case for the specific function (40)).

The calculation shows that

$$G^{**}(\xi,\eta) = |\xi||\eta| \begin{cases} u_+\cos 2\chi, & 0 \leq \tan\chi \leq \sqrt{u_-/u_+}, \\ (u_+ + u_-)\cos^2\chi - \sqrt{u_+u_-}\sin 2\chi, & \\ \qquad\qquad \sqrt{u_-/u_+} \leq \tan\chi \leq \sqrt{u_+/u_-}, & \\ u_-\cos 2\chi, & \sqrt{u_+/u_-} \leq \tan\chi \leq \infty. \end{cases} \tag{44}$$

Here, 2χ denotes the angle formed by the vectors $\xi = \nabla T$ and $\eta = \nabla\lambda$ at the corresponding point (x, y).

It can be checked directly that the inequality (43) holds. We arrive at the inequality

$$\sup_{u,T}\inf_\lambda J \leq \sup_T \inf_\lambda \left[-\int_{-1}^{1}\lambda(x,1)\,dx + \int_S G^{**}(\nabla T, \nabla\lambda)\,dx\,dy\right] \tag{45}$$

which is the required upper bound.

On the other hand, the functional $\sup_{u,T}\inf_\lambda J$ may be estimated from below if we evaluate it for some specific microstructure; let $D_0 = d_1e_1e_1 + d_2e_2e_2$ be the tensor of its effective heat conductances. Assume that D_0 is chosen according to the following rule: (a) for $0 \leq \tan\chi \leq \sqrt{u_-/u_+}$ we set $D_0 = u_+E$ (pure u_+ material), (b) for $\sqrt{u_+/u_-} \leq \tan\chi \leq \infty$ we set $D_0 = u_-E$ (pure u_- material), and (c) for $\sqrt{u_-/u_+} \leq \tan\chi \leq \sqrt{u_+/u_-}$ we apply a layered composite whose tensor D_0 has eigenvalues

$$d_1 = [mu_+^{-1} + (1-m)u_-^{-1}]^{-1}, \tag{46}$$

$$d_2 = mu_+ + (1-m)u_-, \tag{47}$$

and where the eigenvector e_2 bisects the angle 2χ between ∇T and $\nabla\lambda$. The concentration m in the latter case will be chosen in accordance

with the rule

$$u_+ u_- / d_1^2 = \tan^2 \chi. \tag{48}$$

Now we have to use the integrand $\nabla T \cdot D_0 \cdot \nabla \lambda$ instead of $u \nabla T \cdot \nabla \lambda$ in the original setting; we have finally

(49)

$$\nabla T \cdot D_0 \cdot \nabla \lambda = |\nabla T|\,|\nabla \lambda| \begin{cases} u_+ \cos 2\chi, & 0 \le \tan \chi \le \sqrt{u_-/u_+}, \\ (u_+ + u_-)\cos^2 \chi - \sqrt{u_+ u_-} \sin 2\chi, & \\ \qquad\qquad \sqrt{u_-/u_+} \le \tan \chi \le \sqrt{u_+/u_-}, & \\ u_- \cos 2\chi, & \sqrt{u_+/u_-} \le \tan \chi \le \infty. \end{cases}$$

Comparing this with (44) we see that the two bounds, upper and lower, of the functional $\sup_{u,T} \inf_{\lambda} J$ are coincident, which means that the desired supremum is attained and is equal to

$$\max_T \min_\lambda \left[-\int_{-1}^{1} \lambda(x, 1)\, dx + \int_s G^{**}(\nabla T, \nabla \lambda)\, dx\, dy \right]. \tag{50}$$

One can see that the variety of *optimal regimes* offered by (44) does not differ from that provided by (32)–(34), the latter deduced from the explicit formulas (23) for a G-closure. In other words, both procedures lead to the same results when applied to a specific problem of optimization.

Other applications. The direct approach developed here can be applied to a wide class of optimal design problems. In this respect, special mention should be made of elastic rods, plates, and shells as the most widely used constructive elements. For all these, the spatial distribution of materials possessing different values of elastic moduli presents a very effective controller. For plates and shells, a similar role is played by the distribution of thickness along their midsurfaces. The latter problems could equally well be associated with the desire to save as much material as possible. The total cost of material used is also of great importance; a strong material is often more expensive than a weak one, and one may wish to make the combined construction be the strongest of all affordable. There is little physical intuition concerning optimal distributions of materials or thickness along the midsurfaces of plates and shells. The experience already gained (much of it having come directly from engineering practice) shows that such designs are characterized by the formation of microstructures, e.g., grillage—like systems of ribs for

a plate of variable thickness. Structural parameters of these systems, their topology and orientation, should be determined in the course of optimization.

The new approach described above can specifically be applied to these types of problems, and essentially speaking, this is the only approach which may then be suggested, because for most of these problems, G-closures of the typical U-sets are unknown. We could also work with a fairly broad range of cost functionals, among them all functionals which are weakly continuous in the corresponding Sobolev spaces. It is also very important that the dimensionality of spaces to be used in the process of obtaining a solution (be it analytical or numerical) be the same as the dimensionality of the space of dependent variables, not of the invariant space of the elements of G-closures. In the heat problem described above, this distinction was unimportant (we had the two-dimensional space of invariants of a planar D_0-tensor and the two-dimensional space of ∇T-vectors). For the plate problem, however, the difference will be great: the invariant space of D_0-tensors will be five-dimensional, and the space of strain terms e only three-dimensional; for general elasticity the difference will already be striking: eighteen for the D_0-tensor and six for strain.

This reduction in dimensionality makes it possible to take into account a variety of external conditions under which the same constructive elements may be designed to work. Applied to aircraft elements, for example, this idea may permit us to handle, along with the regime of static equilibrium, also the failure of stability (static and dynamic), additional restrictions upon the spectrum of eigenfrequencies, etc. Also, one could allow for the analysis of optimal design of constructive elements with regard to a combination of physical fields, e.g., the fields of stress and temperature, which is particularly important for work in extremal conditions.

Extensions of the theory. There are fundamental mathematical issues which are still unsolved and which are closely related to the new approach. The fact of coincidence of two types of estimates reflects the property of the transformed integrand to be attained with the aid of some specific material microstructure. Mathematically speaking, this is the case when the solution of the sup-inf form exists and coincides with the value of the max-min form. The fundamental problem of general kind which arises is that of necessary and sufficient conditions of exis-

tence of solution to the sup-inf type of variational problem in the case of many independent variables. It is well known that the saddle point type of behavior of the integrand is sufficient for existence [21]. At the same time, the above-mentioned example shows that this property is not necessary and could allow weaker restrictions on the behavior of the integrand. In other words, that means that the class of functions which are good from the point of view of existence is wider than that of saddle functions. The situation which arises here is very similar to that encountered in the analysis of nonconvex minimization problems in many independent variables [22]. Applied to these problems, the method of two-sided estimates has also demonstrated its effectiveness. In the latter context, ordinary convexity of the integrand was sufficient for existence of the minimum but by no means necessary, and the class of functions providing the minimum was wider than that of the convex functions alone. In the absence of convexity the method of estimates worked well: the upper estimate being provided by some microstructure, and the lower estimate being provided by the polyconvexification transformation [22] which played the role of transformation (42). The following observation seems to be remarkable: *convex functions stay unchanged when subjected to the polyconvexification transformation.* The same phenomenon is observed with regard to saddle functions if we subject them to transformation (42). For these reasons, the fact of coincidence of two types of estimates in the sup-inf context seems to be not accidental, but rather it expresses the property of the transformed integrand to satisfy necessary and sufficient conditions of sup-inf type to be attained. Continuing this analogy between this approach and that of quasiconvexification, we could argue that the transformed integrand plays the same role as the quasiconvex envelope of the integrand in the problem of nonconvex minimization.

The problem consists of developing strict mathematical theory of existence of solution of min-max variational problems with nonsaddle integrands in the case of many independent variables. A notion similar to quasiconvexity should be outlined and examined. Connections must be described between this notion and the behavior of the original function with regard to the passage to weak limits of the arguments (a property similar to that of weak lower semicontinuity). This investigation could be carried out in a broader context than merely optimization; the original function is not necessarily produced by some optimal control problem.

Development of computer software. We now consider the computational aspects of the new approach. Formally speaking, the basic operations associated with the transformation (42) include construction of a convex hull of some original set in the space of dependent variables. This set can be fairly arbitrary; for the plate problem, for example, it could be a segment of some smooth curve in three dimensions, the curve being described in analytical terms. The convex hull of this original curve is some body in three-space. The surface of this body is piecewise smooth, but might also possess vertices, edges, etc. All of these types of features allow an immediate interpretation in terms of microstructures associated with the corresponding layout of materials, and for this reason it is important to build up the convex hull in full detail. This construction of the convex hull is the main difficulty to be overcome by the development of software. *More specially, the various types of points on the surface of the convex hull are each to be set in correspondence with composites of various specific microstructures.* The purpose of the overall analysis is to provide an exhaustive classification of all possible cases which might arise in this connection. Formally speaking, this classification will come as a result of solving a purely geometrical problem

$$\sup_{b \in B} b \cdot \eta \tag{51}$$

where η is some unit vector (which plays the role of a parameter) in the space of dependent variables, and B denotes the above-mentioned convex hull. The unit sphere swept out by the η-vectors is to be partitioned into sections associated with various types of points b extremal in the sense of (51). (These points might be smooth, edge points, vertex points, etc.) The classification mentioned above is not immediate, since each of these types of points is connected with some well-specified microstructure. At this point, then, the software to be developed will have computed the supremum over b in Equation (42). The next computational step would be to compute the outer supremum, the supremum over A in (42). Typically, this will be a low-dimensional optimization problem. Note, however, that the computation (51) needs to be carried out for each evaluation of the objective function in this new (outer) optimization problem, making the combined problem computationally intensive. Finally, given the classification of a regime, the computation of the materials layout is a standard problem for which algorithms already exist [23].

The software to be developed will perform the following tasks. Given a curve in three dimensions (either in parametrized or numerical form) the software will render this curve graphically. It will then compute and display the convex hull of this curve. The user will be able to rotate this image in three dimensions in real time, so as to understand the qualitative nature of the hull (and thus the classification of the composite regimes). Also, given a unit direction vector η, the software will display cross sections of the convex hull which are perpendicular to η; the software will also find the extreme values of b for which the cross sections are tangential to the convex hull. Given these values of b, and so likewise knowing the points of intersection p of the cross sections with the convex hull, the software will categorize the points p as being on the smooth surface of the hull, or on an edge, or on a vertex, etc. Next, the software will perform a low-dimension optimization problem, essentially reshaping the convex hull, so as to maximize the values of

$$\sup_{b \in B} b \cdot \eta + A(\xi_1 \eta_2 - \xi_2 \eta_1)$$

as A varies. The software will then carry out this computation as η sweeps out the unit sphere, and so we will have a complete classification of the regimes.

While convex hulls have been studied and used extensively in mathematics, algorithms for actually computing hulls efficiently have not received as much attention, due possibly to a previous lack of practical applications. Computation of convex hulls in two dimensions have been studied, [24], [25], [26], but the problem in three dimensions has only recently gained attention. From a computational standpoint, then, this is the least understood aspect of the total new approach to materials layout. Note that the issue here is not the difficulty of computing the hulls, since there are many intuitive approaches which can be used, but *computing the hulls efficiently*. That is, if we discretize the parametrization of the original curve with N grid points, then naive approaches to computing the convex hull, such as using the geometric definition, have a computational complexity of $O(N^2)$. Typically, N would be quite large, making computation of the hull time consuming when using naive algorithms, and so defeating attempts to generate real-time images, or to perform the optimization (42) in reasonable time.

The other computational aspects of the new approach are better understood. Determining the extremal values for b is a fairly standard problem in nonlinear programming, albeit one still requiring some care in solving. Likewise, the computation of the supremum over A can

be cast in a standard way as a problem in nonlinear programming; this would not take into account, however, any special structure in the outer optimization problem due to the special nature of the objective function (51). Rendering the image of the hull, of course, would involve standard techniques from computer graphics.

Significance of research. In Figures 4, 5, and 6 we illustrate the specific optimization procedures associated with the naive, G-closure, and new approaches, respectively. For practical considerations it is important to obtain some suboptimal layouts; that is, those characterized by some simple types of internal geometry which could be realized in practice. (In this context simple means depending on some finite number of structural parameters.) Once the ideal (optimal) solution is known, we can introduce some kind of cost functional to estimate the difference between the optimal and suboptimal solution. This provides the basis for evaluating the *best approximation to the optimal layout which could be achieved with the aid of available simplified microstructures.*

In summary, we can say that the proposed (new) approach provides a practical (as compared to G-closure) yet systematic (as compared to naive) methodology to handle layout problems arising in applications.

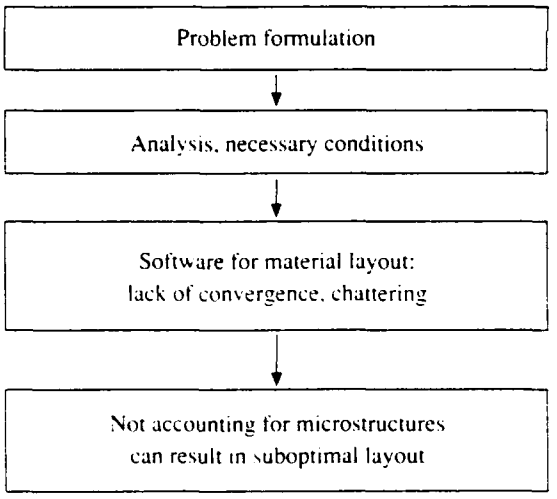


FIGURE 4

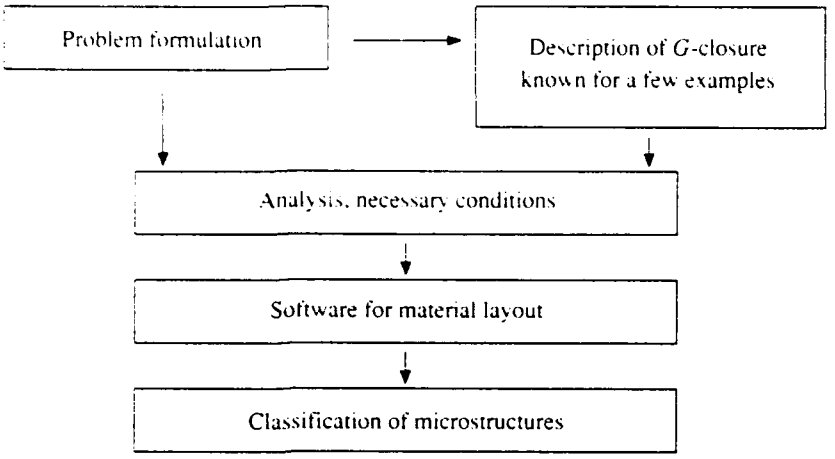


FIGURE 5

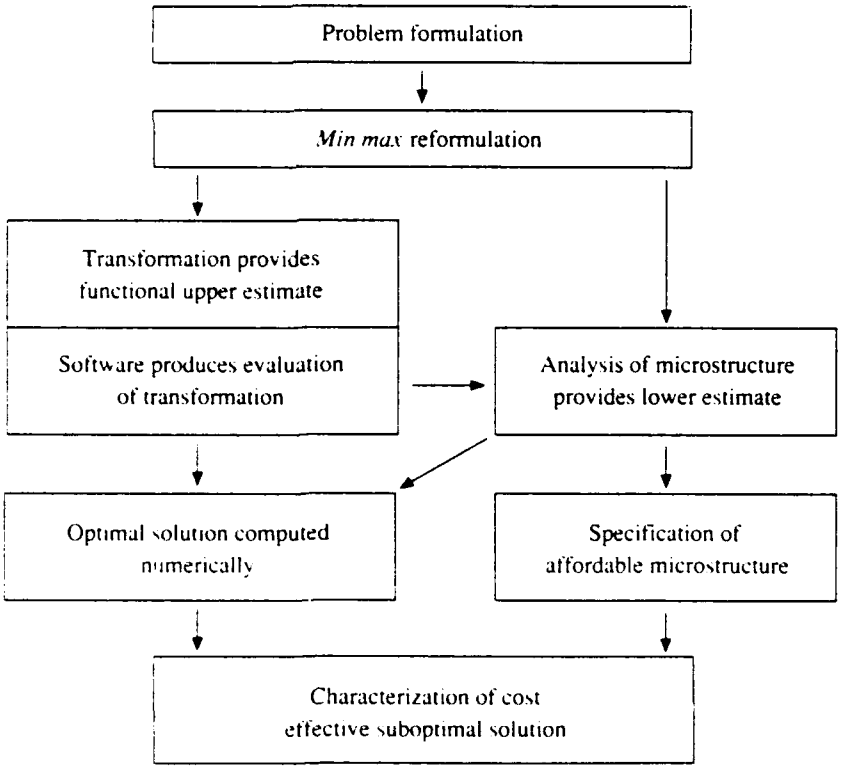


FIGURE 6

Acknowledgment. The author is indebted to James I. Northrup for valuable discussion.

REFERENCES

1. K. A. Lurie, *Optimal Control in Problems of Mathematical Physics* (in Russian), Nauka, Moscow, 1975.

2. J.-L. Armand and B. Lodier, *Optimal design of bending elements*, Internat. J. Numer. Methods Engrg. **13** (1978), 373–384.

3. N. Olhoff and K.-T. Chen, *An investigation concerning optimal design of solid state elastic plates*, Internat. J. Solids and Structures **17** (1981), 305–321.

4. N. A. Lavrov, K. A. Lurie, and A. V. Cherkaev, *Nonhomogeneous rod of extremal torsional rigidity* (in Russian), Izv. Akad. Nauk SSSR, Mekhanika Tverdogo Tela (MTT), **6** (1980).

5. L. C. Young, *Lectures on the calculus of variations and optimal control theory*, Saunders, Philadelphia, 1969.

6. K. A. Lurie and A. V. Cherkaev, *Effective characteristics of composites and optimal structural design* (in Russian), Uspekhi Mekhaniki (Advances in Mechanics) **9** (1986), 3–81.

7. K. A. Lurie, A. V. Fedorov, and A. V. Cherkaev, *Regularization of optimal design problems for bars and plates* (in Russian), Preprint 667, A. F. Ioffe Physical Technical Institute, Acad. Sci. USSR, Leningrad, 1980; see also J. Opt. Theory Appl. **37** (1982), 499–543.

8. K. A. Lurie and A. V. Cherkaev, *G-closure of a set of anisotropic conducting media in the two-dimensional case*, The Danish Center for Applied Math. and Mech. Reprint 213, 1981; see also J. Opt. Theory Appl. **42** (1984), 283–304, corr. **53** (1987), 319.

9. ——, *G-closure of some particular sets of admissible material characteristics for the problem of bending of thin plates*, The Danish Center for Applied Math. and Mech. Reprint 214, 1981; see also J. Opt. Theory Appl. **42** (1984), 305–315.

10. ——, *Exact estimates of conductivity of composites formed by two isotropically conducting media taken in prescribed proportion*, Proc. Roy. Soc. Edinburgh **99A** (1984), 17.

11. ——, *Exact estimates of the conductivity of a binary mixture of isotropic materials*, Proc. Roy. Soc. Edinburgh **104A** (1986), 21–28.

12. L. Tartar, *Problèmes de contrôle des coefficients dans les équations aux derivées partielles*, in Control Theory, Numerical Methods, and Computer System Modelling (A. Bensoussan and J.-L. Lions, eds.), Lecture Notes in Economics and Math. Systems **107** (1975), 420–426.

13. ——, *Estimations fines des coefficients homogeneises*, in Ennio DeGiorgi's Colloquium (P. Kree, ed.), Research Notes in Mathematics, Pitman Press, London, 1985, pp. 168–187.

14. U. E. Raitum, *The extension of extremal problems connected with a linear elliptic equation*, Soviet Math. Dokl. **19** (1978), 1342–1345.

15. G. A. Francfort and F. Murat, *Homogenization and optimal bounds in linear elasticity*, Arch. Rat. Mech. Anal. **94** (1986), 307.

16. ——, *Optimal bounds for conduction in two-dimensional, two-phase media*, in Proceedings of the Durham symposium on nonclassical continuum mechanics (R. J. Knops, ed.), Cambridge University Press, Cambridge, 1987.

17. M. Avellaneda, *Optimal bounds and microgeometries for elastic composites*, SIAM J. Appl. Math. **47** (1987), 1216.

18. G. W. Milton, *Modelling the properties of composites by laminates*, in Homogenization and effective moduli of materials and media (J. L. Ericksen et al., eds.), Springer-Verlag, New York, 1986.

19. G. W. Milton and R. V. Kohn, *Variational bounds on the effective moduli of anisotropic composites*, J. Mech. Phys. Solids **36** (1988), 597–629.

20. K. A. Lurie, *The extension of optimization problems containing controls in the coefficients*, Proc. Roy. Soc. of Edinburgh **114A** (1990), 81–97.

21. R. Rockafellar, *Convex Analysis*, Princeton University Press, Princeton, N.J., 1970.

22. R. V. Kohn and G. Strang, *Optimal design and relaxation of variational problems*, Comm. Pure Appl. Math. **39** (1986), 113, 139, 353.

23. N. Banichuk, *Problems and Methods of Optimal Structural Design*, Plenum Press, New York, and London, 1983.

24. R. L. Graham, *An efficient algorithm for determining the convex hull of a finite planar set*, Inform. Process. Lett. **1** (1972), 132–133.

25. ——, *Finding the convex hull of a simple polygon*, J. Algorithms **4** (1983), 324–331.

26. A. A. Schaffer and C. J. VanWyk, *Convex hulls of piecewise-smooth Jordan curves*, J. Algorithms **8** (1987), 66–94.

Department of Mathematical Sciences, Worcester Polytechnic Institute, Worcester, MA 01609-2280

E-mail address: KLURIE@WPI

In 'Theoretical Aspects of Industrial Design', ed by David A. Field & Vadim Komkov, SIAM, 1992, pp 1-11

DIRECT SOLUTION OF AN OPTIMAL LAYOUT PROBLEM FOR ISOTROPIC HEAT CONDUCTORS IN THREE DIMENSIONS*

K. A. LURIE† AND R. LIPTON††

Abstract. The paper suggests a procedure of direct construction of minimal extension of constrained optimization problem for a three dimensional heat equation containing controls in coefficients. For a two dimensional case, this approach has been initiated in Ref. 1.

1. Introduction. We consider nonselfadjoint optimization problems for a system of equation in a three dimensional region V

$$q = \mathcal{D} \cdot \nabla T, \quad \nabla \cdot q = 0 \tag{1}$$

where $T = T(x,y,z)$ denotes the temperature and the tensor $\mathcal{D} = \mathcal{D}(x,y,z)$ of heat conductance plays the role of control. The set U of admissible values of $\mathcal{D}$ includes two elements (materials):

$$U = \{\mathcal{D} : \mathcal{D}_+ = u_+ E, \mathcal{D}_- = u_- E, E = ii + jj + kk\}. \tag{2}$$

It is required to find the distribution

$$\mathcal{D}(x,y,z) = \chi_1(x,y,z)\mathcal{D}_+ + \chi_2(x,y,z)\mathcal{D}_-$$

of a heat conductance tensor throughout V which maximizes some weakly continuous functional $I(T)$. Here T denotes a solution to the boundary–value problem obtained when Eqs. (1) are complemented by the linear boundary condition

*The work of KAL has been supported by AFOSR Grant No. 90–0268. This support is kindly appreciated.

†Professor, Department of Mathematical Sciences, Worcester Polytechnic Institute, Worcester, MA 01609.

††Assistant Professor, Department of Mathematical Sciences, Worcester Polytechnic Institute, Worcester, MA 01609.

$$L(T) = f \tag{3}$$

along ∂V. To specify the problem, we will consider the Dirichlet condition

$$T\Big|_{\partial V} = f \tag{4}$$

and the functional

$$I(T) = -\int_V [T(x,y) - T_0(x,y)]^2 dxdydz, \tag{5}$$

where $T_0(x,y) \in L_2(V)$.

This problem is known to be ill–posed and therefore requiring relaxation, i.e., the construction of an appropriate minimal extension of the initial set U of admissible controls. Such an extension can be constructed on the basis of a precise knowledge of the G–closure of U, i.e., the set GU of invariants of the effective heat conductance tensors $\mathcal{D}_C$ of all composites assembled from the elements of U (Ref. 2). However, the G–closures are known only for a few particular examples, the case of U defined by (2) among them (Ref. 3). Yet for these selected examples, the construction of GU represents a difficult problem. For more complex situations, e.g., that of an elliptic equation of the 4th order, the problem of constructing the G–closure still remains open.

At the same time, for many applications we do not need to know the GU–set in full detail. Instead, it is often enough to specify some linear combinations of components of $\mathcal{D}_0$, particularly, for our example the combination $\mathcal{D}_0 \cdot \nabla T$ entering the first of equations (1). This is the only combination which really matters for our purposes; to determine it, we apply a *direct approach*, free from any reference to the G–closure.

A similar problem in two dimensions has been discussed in Ref. 1.

2. Transformation of the problem. We first reduce the problem to the convenient sup inf form. Introduce the Lagrange multiplier λ and consider the augmented functional

$$J = J(T,\lambda) = I(T) - \int_V \lambda \nabla \cdot \mathcal{D} \cdot \nabla T \, dxdydz, \tag{6}$$

the right–hand side taking into account the heat equation

$$\nabla \cdot \mathcal{D} \cdot \nabla T = 0 \tag{7}$$

following from (1).

Equating to zero the first variation of (6) with respect to T and bearing (4) in mind, we arrive at the conjugate equation

$$\nabla \cdot \mathcal{D} \cdot \nabla \lambda = -2(T - T_0) \tag{8}$$

and the boundary condition

$$\lambda\Big|_{\partial V} = 0. \tag{9}$$

After integration by parts with the boundary condition (9), the functional (6) takes on the form

$$J = I(T) + \int_V \nabla\lambda \cdot \mathcal{D} \cdot \nabla T \, dxdydz \tag{10}$$

convenient for subsequent use.

It can be shown that the problem

$$\sup_{\mathcal{D}} I,$$

subjected to (1) and (4) is equivalent to

$$\sup_{\mathcal{D},T} \inf_{\lambda} J$$

subjected to (4), (9). Indeed, since

$$J = I(T) + \int_{\partial V} \lambda n \cdot \mathcal{D} \cdot \nabla T dS - \int_V \lambda \nabla \cdot \mathcal{D} \cdot \nabla T \, dxdydz$$

the operation $\inf_{\lambda} J$ yields, in view of (9),

$$\inf_{\lambda} J = I(T),$$

the constraint (7) now appearing as a necessary condition for a minimum in λ.

The functional $\sup_{\mathcal{D},T} \inf_{\lambda} J$ has the following upper bound:

$$\sup_{\mathcal{D},T} \inf_{\lambda} J = \sup_{T} \sup_{\mathcal{D}} \inf_{\lambda} J \leq \sup_{T} \inf_{\lambda} \sup_{\mathcal{D}} J$$

$$= \sup_{T} \inf_{\lambda} \left[- \int_V (T - T_0)^2 dxdydz + \int_V G(\nabla T, \nabla \lambda) dxdydz \right] \qquad (11)$$

where (we accept the notation $\xi = \nabla T$, $\eta = \nabla \lambda$)

$$G(\xi,\eta) = \begin{cases} u_+ \xi \cdot \eta, & \text{if } \xi \cdot \eta \geq 0, \\ u_- \xi \cdot \eta, & \text{if } \xi \cdot \eta \leq 0. \end{cases} \qquad (12)$$

The function $G(\xi,\eta)$ is convex with respect to any of its arguments but non–convex with respect to their union.

The problem

$$\sup_{T} \inf_{\lambda} \left[- \int_V (T - T_0)^2 dxdydz + \int_V G(\nabla T, \nabla \lambda)) dxdydz \right], \; T \in (4), \; \lambda \in (9) \qquad (13)$$

is still ill–posed. It would be well–posed if the integrand $G(\xi,\eta)$ were a saddle function, i.e. concave in ξ for fixed η and convex in η for fixed ξ. The solution would then exist and the operations sup and inf would commute. For our problem it is obviously not the case. However, the requirement that the function $G(\xi,\eta)$ be saddle is too restrictive when ξ and η are gradients; to ensure existence of sup inf for this case it is enough to require that this function be only quasisaddle (Ref. 1). We will consider in this connection its quasisaddle envelope $G^{**}(\xi,\eta)$ applying the s.c. polysaddlification transformation introduced in Ref. 1 and playing the same role in sup inf problems as the polyconvexification transformation (Ref. 4–6) plays for the infimum problems. This new transformation is given by the formula

$$G^{**}(\xi,\eta) = \sup_{\omega} \sup_{b} \inf_{a} \{ a \cdot \xi + b \cdot \eta + \omega \cdot \xi \times \eta$$

$$- \inf_{\xi} \sup_{\eta} [a \cdot \xi + b \cdot \eta + \omega \cdot \xi \times \eta - G(\xi,\eta)] \}. \qquad (14)$$

The term $\omega \cdot \xi \times \eta$ represents the null–Lagrangians $(\xi \times \eta)_1 = \eta_2 \xi_3 - \eta_3 \xi_2$, $(\xi \times \eta)_2 = \eta_3 \xi_1 - \eta_1 \xi_3$, $(\xi \times \eta)_3 = \eta_1 \xi_2 - \eta_2 \xi_1$ taken into account with the aid of Lagrange multiplier $\omega(\omega_1, \omega_2, \omega_3)$. If $G(\xi,\eta)$ is convex in η (which is now the case, see (12)) and arbitrary in ξ, then (Ref. 1)

$$G^{**}(\xi,\eta) \geq G(\xi,\eta). \qquad (15)$$

This inequality represents the characteristic property of $G^{**}(\xi,\eta)$ making it possible to use this function instead of $G(\xi,\eta)$ in (13) and thus arrive at the upper bound for this functional.

3. Computation of $G^{**}(\xi,\eta)$: the upper bound for sup inf J. We first compute $\overline{h}(\xi,b) = \sup_{\eta} [b \cdot \eta - H(\xi,\eta)]$ with $H(\xi,\eta) = -\omega \cdot \xi \times \eta + G(\xi,\eta)$. This computation is similar to that of the two dimensional analysis done in Ref. 1. We obtain

$$h(\xi,b) = \sup_{\eta}[b\cdot\eta - H(\xi,\eta)] = \begin{cases} 0 \quad \text{if} \quad b + \omega \times \xi - u\xi = 0, \ u_{-} \le u \le u_{+}, \\ +\infty \ \text{otherwise} . \end{cases} \tag{16}$$

The transformation (14) now involves the operation

$$\inf_{a}\{a \cdot \xi - \inf_{\xi}[a \cdot \xi - (-\overline{h}(\xi,b))]\} \tag{17}$$

which yields the concave envelope of $-\overline{h}(\xi,b)$ with respect to the $\xi-$variable for fixed b.

According to (16), $-\overline{h}(\xi,b) = 0$ along the arc $u \in [u_{-},u_{+}]$ of the curve γ in the ξ–space

$$\gamma: \ b = -\omega \times \xi + u\xi = S(u) \cdot \xi, \tag{18}$$

or, explicitly in terms of ξ,

$$\xi = S^{-1}(u) \cdot b, \ u_{-} \le u \le u_{+} < \infty,$$

$$S^{-1}(u) \cdot b = \frac{u}{u^2+\omega^2} b + \frac{b \cdot \omega}{u(u^2+\omega^2)} \omega + \frac{\omega \times b}{u^2+\omega^2}. \tag{19}$$

If we introduce a Cartesian coordinate system (x,y,z) with z–axis parallel to the ω–vector ($\omega = 0,0,\omega$)), then (19) will reduce to the system

$$\xi_x = \frac{u}{u^2+\omega^2} b_x - \frac{\omega}{u^2+\omega^2} b_y,$$

$$\xi_y = \frac{u}{u^2+\omega^2} b_y + \frac{\omega}{u^2+\omega^2} b_x, \tag{20}$$

$$\xi_z = \frac{1}{u} b_z ,$$

showing that the curve (18) lies on the surface of the cylinder (Fig. 1)

$$\omega(\xi_x^2 + \xi_y^2) + b_y\xi_x - b_x\xi_y = 0. \tag{21}$$

Figure 1. The shape of curve (19)

The concave envelope (17) will now be defined as

$$\inf_{a}\{a \cdot \xi - \inf_{\xi}[a \cdot \xi - (-\overline{h}(\xi,b))]\} = \begin{cases} 0\ , & \xi \in \Xi \\ \\ -\infty, & \xi \notin \Xi \end{cases} \tag{22}$$

where Ξ is the convex hull of the curvilinear segment (19).

The hull is a convex body with boundary composed of two sheets. These sheets intersect along the arc γ with the endpoints A and B given by

$$\begin{aligned} &A:\ \xi = S^{-1}(u_+) \cdot b = S_+^{-1} \cdot b \\ \\ &B:\ \xi = S^{-1}(u_-) \cdot b = S_-^{-1} \cdot b, \end{aligned} \tag{23}$$

respectively. Also, they intersect along the straight line segment (chord) AB

$$\xi = tS_+^{-1} \cdot b + (1-t)S_-^{-1} \cdot b,\ t \in [0,1] \tag{24}$$

connecting the same points. With the reference to Fig. 1 and because u_+ is finite, it is obvious that both the arc γ and the chord AB belong to the boundary of the convex hull Ξ.

In view of the subsequent $\sup_{b}$ operation in (14), we have to interpret (22) as the function depending on the argument b for fixed ξ. It is remarkable that Eq. (19) which represents an arc of γ in ξ–space may be interpreted as Eq. (18) representing a straight line $\overline{\gamma}$ in b–space. Analogously, Eq. (24) represents a chord in ξ–space and at the same time a curve $\overline{AB}$ in b–space. Both will belong to the boundary of a body $\mathcal{B}$ in b–space which appears as we interpret the lefthand side of Eq. (22) as a function of b for fixed ξ:

$$\inf_{a}\{a \cdot \xi - \inf_{\xi}[a \cdot \xi - (-\overline{h}(\xi,b))]\} = \begin{cases} 0\ , & b \in \mathcal{B} \\ \\ -\infty, & b \notin \mathcal{B} \end{cases} \tag{25}$$

In view of (25), the operation

$$\sup_{b}\{b \cdot \eta - \inf_{a}[a \cdot \xi - \inf_{\xi}(a \cdot \xi - (-\overline{h}(\xi,b)))]\}$$

reduces to

$$\sup_{b \in \mathcal{B}} b \cdot \eta$$

or

$$\sup_{b \in \operatorname{conv}\mathcal{B}} b \cdot \eta\ . \tag{26}$$

Because the chord (18) and the curve (24) obviously belong to the boundary of the convex hull conv $\mathcal{B}$ of $\mathcal{B}$, they should be tested for optimality in terms of the operation (26). Since the body conv $\mathcal{B}$ in convex, its tangent planes participating in the computation of supremum will touch its surface, among other points, also either at its vertices $\overline{A}$ and $\overline{B}$ or along the arc $\overline{AB}$. We first consider the case where the contact point occurs on $\overline{AB}$ and carry out the combined computation for

$$\sup_{\omega} \sup_{b \in \overline{AB}} [b \cdot \eta + \omega \cdot \xi \times \eta].$$

Though this expression is generally less than $G^{**}(\xi,\eta)$, we expect that it will nevertheless satisfy the inequality

$$\sup_{\omega} \sup_{b \epsilon \overline{AB}} [b \cdot \eta + \omega \cdot \xi \times \eta] \geq G(\xi,\eta)$$

and thus provide us with the upper estimate for $G(\xi,\eta)$.

In Eq. (24) for $\overline{AB}$, the matrices S_+ and S_- are defined by (23) as $S(u_+)$, $S(u_-)$, respectively, where $S(u)$ is given by (see(18))

$$S(u) = \omega \cdot \epsilon + uE \tag{27}$$

where $E = ii + jj + kk$ is a unit tensor, and $\epsilon = -E \times E$ is the Levi–Civita tensor of the third rank. The combination

$$tS_+^{-1} \cdot b + (1-t)S_-^{-1} \cdot b,$$

appearing in (24) represents the ξ–convexification of the basepoints $S_+^{-1} \cdot b$, $S_-^{-1} \cdot b$ of the curve γ (see(18)) corresponding to the values u_+, u_- of parameter u. The operation

$$\sup_{\omega} \sup_{b \epsilon \overline{AB}} [b \cdot \eta + \omega \cdot \xi \times \eta]$$

now reduces to the examination of the extreme points of the function

$$K = K(\omega,b,t) = b \cdot \eta + \Lambda \cdot [\xi - tS_+^{-1} \cdot b - (1-t)S_-^{-1} \cdot b] + \omega \cdot \xi \times \eta \tag{28}$$

where Λ denotes the Lagrange multiplier for the constraint (24). The necessary conditions for extrema require that

$$K_b = \eta - t\Lambda \cdot S_+^{-1} - (1-t)\Lambda \cdot S_-^{-1} = 0, \tag{29}$$

$$K_\omega = -t(1-t)[(S_+^{-1} - S_-^{-1}) \cdot b] \times \{\Lambda \cdot (S_+^{-1} - S_-^{-1})] = 0. \tag{30}$$

$$K_t = -\Lambda \cdot (S_+^{-1} - S_-^{-1}) \cdot b = 0. \tag{31}$$

The Eqs. (29) and (31) are obvious whereas Eq. ((30) follows from the analysis given in Appendix 1.

Eq. (30) shows that

$$(S_+^{-1} - S_-^{-1}) \cdot b = \alpha\Lambda \cdot (S_+^{-1} - S_-^{-1}) \tag{32}$$

where α is a scalar multiplier.

Introducing the symbols m,n,p defined as

$$m = \frac{u_+}{u_+^2+\omega^2} - \frac{u_-}{u_-^2+\omega^2},$$

$$n = \frac{1}{u_+(u_+^2+\omega^2)} - \frac{1}{u_-(u_-^2+\omega^2)}, \tag{33}$$

$$p = \frac{1}{u_+^2+\omega^2} - \frac{1}{u_-^2+\omega^2},$$

we obtain

$$S_+^{-1} - S_-^{-1} = mE + n\omega\omega - p\omega \cdot \epsilon,$$

and Eqs. (32) and (31) become

$$mb + n(b \cdot \omega)\omega + p\omega \times b = \alpha[m\Lambda + n(\Lambda \cdot \omega)\omega - p\omega \times \Lambda], \tag{34}$$

$$m\Lambda \cdot b + n(\Lambda \cdot \omega)(b \cdot \omega) + p\Lambda \cdot \omega \times b = 0, \tag{35}$$

respectively.

This system can be simplified. To this end we assume that neither of three vectors

$$\omega \times b, \ \omega \times \Lambda, \ \Lambda \times b$$

is zero. Computing the dot products of (34) with b, Λ ω respectively, we arrive at

$$mb^2 + n(b \cdot \omega)^2 = \alpha[m\Lambda \cdot b + n(\Lambda \cdot \omega)(b \quad \omega) - pb \cdot \omega \times \Lambda], \tag{36}$$

$$m\Lambda \cdot b + n(\Lambda \cdot \omega)(b \cdot \omega) + pb \cdot \Lambda \times \omega = \alpha[m\Lambda^2 + n(\Lambda \cdot \omega)^2], \tag{37}$$

$$(b \cdot \omega - \alpha\Lambda \cdot \omega)(m + n\omega^2) = 0\,. \tag{38}$$

Assuming that $\alpha \neq 0$ and taking (35) into account, we conclude that Eqs. (36) and (37) reduce to

$$m\Lambda^2 + n(\Lambda \cdot \omega)^2 = 0, \tag{39}$$

$$mb^2 + n(b \cdot \omega)^2 = 0. \tag{40}$$

Eq. (38) allows for two possibilities: $m + n\omega^2 = 0$ and $b \cdot \omega = \alpha\Lambda \cdot \omega$. It is evident from (33) that the first never occurs. As to the second possibility, this together with (39) and (40) implies

$$m(b^2 - \alpha^2\Lambda^2) = 0,$$

which means that either $m = 0$ or $|b| = |\alpha||\Lambda|$. The first possibility together with (39) and (40) implies that

$$\Lambda \cdot \omega = b \cdot \omega = 0 \tag{41}$$

which in view of (35) shows that

$$\Lambda = \beta b \tag{42}$$

where β is a scalar multiplier. We thus conclude that $\Lambda \times b = 0$ which is admissible as a final result.

REMARK: Eqs. (41) imply that the curve (24) in the b–space lies in the plane perpendicular to ω.

As to the second alternative $|b| = |\alpha||\Lambda|$, this one together with $b \cdot \omega = \alpha\Lambda \cdot \omega$ does not contradict the Eqs. (35)–(37) of stationarity. This possibility is in fact eliminated as we demand that the Legendre condition $K_{\omega\omega} \leq 0$ holds. This is shown in Appendix 2. There it is also demonstrated that the relationships (41) together with $m = 0$ satisfy the Legendre condition provided that $\beta \geq 0$.

The requirement $m = 0$ yields

$$\omega^2 = u_+ u_-\,. \tag{43}$$

Now, in view of (27), (33), (42), (43), Eqs. (24) and (29) can be rewritten as follows

$$\xi = \frac{1}{u_+ + u_-}\, b + \frac{1}{(u_+ + u_-)d_1}\, \omega \times b,$$

$$\eta = \beta\left[\frac{1}{u_+ + u_-}\, b - \frac{1}{(u_+ + u_-)d_1}\, \omega \times b\right], \tag{44}$$

where

$$d_1 = \frac{u_+ u_-}{tu_- + (1-t)u_+},$$

and the vector $b \epsilon \overline{AB}$ maximizing the function $K(\omega, b, t)$ is equal to

$$b = \frac{u_+ + u_-}{2}\left[\xi + \frac{|\xi|}{|\eta|}\,\eta\right]$$

which coincides with the result obtained in Ref. 1 for the two dimensional case. We then arrive at the expression

$$(u_+ + u_-)\cos^2\chi - \sqrt{u_+ u_-}\,\sin 2\chi$$

for $\sup_{\omega} \sup_{b\epsilon\overline{AB}}[b \cdot \eta + \omega \cdot \xi \times \eta]$; here, 2χ denotes the angle between vectors ξ and η. This result is valid provided that $\sup_{b\epsilon\overline{AB}}$ occurs at some point b within the arc $\overline{AB}$. This is equivalent to the requirement

$$\sqrt{\frac{u_-}{u_+}} \le \tan\chi \le \sqrt{\frac{u_+}{u_-}}.$$

Another two possibilities allow for $\sup_{b\epsilon\overline{AB}}$ to be attained at the endpoints $\overline{A}$ or $\overline{B}$. We finally obtain (Ref. 1)

$$\sup_{\omega} \sup_{b\epsilon\overline{AB}}[b\cdot\eta+\omega\cdot\xi\times\eta]=|\xi||\eta|\begin{cases} u_+\cos 2\chi, & 0 \le \tan \le \sqrt{\frac{u_-}{u_+}}, \\ (u_++u_-)\cos^2\chi-\sqrt{u_+u_-}\sin 2\chi, & \sqrt{\frac{u_-}{u_+}}\le\tan\chi\le\sqrt{\frac{u_+}{u_-}}, \\ u_-\cos 2\chi, & \sqrt{\frac{u_+}{u_-}} \le \tan\chi \le \infty. \end{cases} \tag{45}$$

The expression (45) is an upper bound for (12) which can be checked as in Ref. 1. On the other hand, this expression can be achieved by rank 1 laminate. To illustrate that, we introduce at each point the plane spanned by vectors ξ,η, with normal ω. Then we introduce a layered composite with layers aligned parallel to ω and bisecting the angle 2χ between ξ and η. For this microstructure, with its effective tensor $\mathscr{D}_0$ and the concentration t chosen in accordance with the rule (Ref. 1)

$$\frac{u_+u_-}{d_1^2} = \tan^2\chi,$$

the expression for $\xi \cdot \mathscr{D}_0 \cdot \eta$ coincides with that of (45) and thus proves its attainability.

4. Conclusion. We thus arrive at the conclusion which has to be expected: the three dimensional case is essentially the same as its two dimensional counterpart. At each point the optimal layout is that which occurs within the local plane defined by vectors ξ and η.

5. Appendix 1: Derivation of Eq. (30)

The ω–derivative of a scalar function $-\Lambda \cdot S^{-1} \cdot b$ is computed in accordance with the rule

$$(-\Lambda \cdot S^{-1} \cdot b)_\omega = [((-\Lambda \cdot S^{-1})_\omega \cdot r_s) \cdot b]r^s = -[(\Lambda \cdot (S^{-1})_\omega \cdot r_s) \cdot b]r^s$$

where r^s is a vector basis (Ref. 7).

The ω–dependence of the tensor S^{-1} will be perceived occurring through the dependence of this tensor on $S(\omega)$. We obtain $(C_{II}=r_sr_tr^sr^t)$

$$\delta S^{-1} = (S^{-1})_\omega \cdot \delta\omega^T = (S^{-1})_S \cdot\cdot \delta S^T;$$

$$\delta S = S_\omega \cdot \delta\omega^T;\ \delta S^T = C_{II} \cdot\cdot \delta S = C_{II} \cdot\cdot S_\omega \cdot \delta\omega^T;$$

$$\delta S^{-1} = (S^{-1})_S \cdot\cdot C_{II} \cdot\cdot S_\omega \cdot \delta\omega^T,$$

and, consequently,

$$(S^{-1})_\omega = (S^{-1})_S \cdot\cdot C_{II} \cdot\cdot S_\omega = (S^{-1})_S \cdot\cdot r_s r_t (r^s r^t \cdot\cdot S_\omega).$$

On the other hand,

$$(S^{-1})_S = -S^{-1} \cdot r_t r_s \cdot S^{-1} r^t r^s$$

and

$$(S^{-1})_S \cdot\cdot r_\alpha r_\beta = -S^{-1} \cdot r_\beta r_\alpha \cdot S^{-1},$$
$$(S^{-1})_\omega = -S^{-1} \cdot r_\beta r_\alpha \cdot S^{-1} (r^\alpha r^\beta \cdot\cdot S_\omega).$$

But $S_\omega = \epsilon = \epsilon^{pqr} r_p r_q r_r$, and consequently

$$r^\alpha r^\beta \cdot\cdot S_\omega = r^\alpha r^\beta \cdot\cdot \epsilon^{pqr} r_p r_q r_r = \epsilon^{\alpha\beta r} r_r.$$

We obtain

$$(S^{-1})_\omega = -S^{-1} \cdot r_\beta r_\alpha \cdot S^{-1} (\epsilon^{\alpha\beta r} r_r) = -S^{-1} \cdot (r_1 r_2 - r_2 r_1) \cdot S^{-1} r_3$$
$$- S^{-1} \cdot (r_2 r_3 - r_3 r_2) \cdot S^{-1} r_1 - S^{-1} \cdot (r_3 r_1 - r_1 r_3) \cdot S^{-1} r_2.$$

As we compute the expression

$$-[(\Lambda \cdot (S^{-1})_\omega \cdot r_s) \cdot b] r^s,$$

assume that the basis r_s is orthonormal; we obtain, for example,

$$-[(\Lambda \cdot (S^{-1})_\omega \cdot r_3) \cdot b] r^3 = [\Lambda \cdot S^{-1} \cdot (r_1 r_2 - r_2 r_1) \cdot S^{-1} (r_3 \cdot r_3) \cdot b] r^3$$
$$= [\Lambda \cdot S^{-1} \cdot (r_1 r_2 - r_2 r_1) \cdot S^{-1} \cdot b] r^3,$$

and analogous expressions for other components. We finally arrive at the formula

$$-(\Lambda \cdot S^{-1} \cdot b)_\omega = -[(\Lambda \cdot (S^{-1})_\omega \cdot r_s) \cdot b] r^s = \Lambda \cdot S^{-1} \times S^{-1} \cdot b.$$

We now obtain

$$K_\omega = -t(\Lambda \cdot S_+^{-1} \cdot b)_\omega - (1-t)(\Lambda \cdot S_-^{-1} \cdot b)_\omega + \xi \times \eta$$
$$= t\Lambda \cdot S_+^{-1} \times S_+^{-1} \cdot b + (1-t)\Lambda \cdot S_-^{-1} \times S_-^{-1} \cdot b +$$
$$+[tS_+^{-1} \cdot b + (1-t)S_-^{-1} \cdot b] \times [t\Lambda \cdot S_+^{-1} + (1-t)\Lambda \cdot S_-^{-1}] =$$
$$= (-t + t^2) S_+^{-1} \cdot b \times \Lambda \cdot S_+^{-1} + (t-t^2) S_-^{-1} \cdot b \times \Lambda \cdot S_+^{-1} +$$
$$+[-(1-t) + (1-t)^2] S_-^{-1} \cdot b \times \Lambda \cdot S_-^{-1} + (t-t^2) S_+^{-1} \cdot b \times \Lambda \cdot S_-^{-1} =$$
$$= -t(1-t)(S_+^{-1} - S_-^{-1}) \cdot b \times \Lambda \cdot (S_+^{-1} - S_-^{-1}).$$

6. Appendix 2: The Legendre condition. The function $K(\omega,b,t)$ defined by (28) should be maximum with respect to ω. This means that $K_{\omega\omega}$ should be negative definite. We apply this test first to the case $m = 0$, $\Lambda = \beta b$, $\Lambda \cdot \omega = b \cdot \omega = 0$, to show that this case is an admissible alternative. The expression for $K_{\omega\omega}$ can be obtained from that for K_ω which is given by (see (32) and (34))

$$K_\omega = -t(1-t)[mb + n(b \cdot \omega)\omega + p\omega \times b] \times [m\Lambda + n(\Lambda \cdot \omega)\omega -$$
$$-p\omega \times \Lambda] = -t(1-t)\{m^2 b \times \Lambda + mn(\Lambda \cdot \omega)(b \times \omega) -$$
$$- pm\, b \times \omega \times \Lambda + mn(b \cdot \omega)(\omega \times \Lambda) - np(b \cdot \omega)\omega \times \omega \times \Lambda -$$

$$-mp\Lambda \times \omega \times b - np(\Lambda \cdot \omega)\omega \times \omega \times b - p^2(\omega \times b) \times (\omega \times \Lambda)\}.$$

Before we differentiate this with respect to ω, we disregard all terms involving more than one factor that vanishes under the hypotheses $m = 0$, $\Lambda = \beta b$ $\Lambda \cdot \omega = b \cdot \omega = 0$. The expression for K_ω then becomes

$$K_\omega = -t(1-t)[-2pm(\Lambda \cdot b)\omega + np(b \cdot \omega)\omega^2\Lambda + np(\Lambda \cdot \omega)\omega^2 b].$$

Differentiation with respect to ω of K_ω gives

$$K_{\omega\omega} = -t(1-t)p[-2(\Lambda \cdot b)m_{\omega^2}\omega\omega + n\omega^2(b\Lambda + \Lambda b)].$$

Here we omitted terms which vanish under the adopted hypotheses. Applying the relationship $\Lambda = \beta b$ and referring to (33) we conclude that

$$K_{\omega\omega} = -t(1-t)p\beta[-2\frac{dm}{d\omega^2}b^2\omega\omega + 2n\omega^2 bb]$$

is negative for $\beta > 0$ because of the inequalities $p < 0$, $n < 0$, $dm/d\omega^2 > 0$ and recalling that $u_+ > u_-$. We now show that the second option, namely $|b| = |\alpha||\Lambda|$, $b \cdot \omega = \alpha\Lambda \cdot \omega$, contradicts the Legendre condition. To this end, we differentiate K_ω with respect to ω, the derivative can be written in the following form:

$$K_{\omega\omega} = -t(1-t)\varphi,$$

$$\varphi = mn[\overset{b}{\Omega} \cdot \omega)\circledast\Lambda + (\Lambda \cdot \omega)\overset{b}{\Omega} - (\overset{\Lambda}{\Omega} \cdot \omega)\circledast b - (b \cdot \omega)\overset{\Lambda}{\Omega}] -$$
$$-pm[(\Lambda \cdot b)E - \Lambda\circledast b] - pm[(\Lambda \cdot b)E - b \circledast \Lambda] -$$
$$-pn\{[\omega(\Lambda \cdot \omega) - \omega^2\Lambda]\circledast b + (b \cdot \omega)[\omega \circledast \Lambda + (\Lambda \cdot \omega)E - 2\Lambda \circledast \omega] +$$
$$+ [\omega(b \cdot \omega) - \omega^2 b] \circledast \Lambda + (\Lambda \cdot \omega)[\omega \circledast b + (b \cdot \omega)E - 2b \circledast \omega]\} +$$
$$p^2[\omega \circledast (\Lambda \times b) + \omega \cdot (\Lambda \times b)E]$$

where $\overset{b}{\Omega}$, $\overset{\Lambda}{\Omega}$ are defined by

$$\overset{b}{\Omega} \cdot \omega = b \times \omega, \quad \overset{\Lambda}{\Omega} \cdot \omega = \Lambda \times \omega,$$

and the symbol $\circledast$ denotes the dyadic product.

We examine the quadratic form $\Omega \cdot K_{\omega\omega} \cdot \Omega$ for the admissible test vector $\Omega = k\,\Lambda \times b$. Referring to (35), it is easy to verify that the form vanishes for this choice. Now, as we perturb the test vector by a small term Ω' and linearize the expression for $(\Omega + \Omega') \cdot K_{\omega\omega'} \cdot (\Omega + \Omega')$ with respect to Ω', then we find the principal part of it to be linear in Ω' which means that it may be made of arbitrary sign for an appropriate choice of Ω'. This completes the proof.

REFERENCES

1. K.A. LURIE, The Extension of Optimization Problems Containing Controls in the Coefficients, Proceedings of the Royal Society of Edinburgh, Vol. 114A, pp. 81–97, 1990.

2. K.A. LURIE, A.V. FEDOROV and A.V. CHERKAEV, Regularization of Optimal Design Problems for Bars and Plates, Parts 1 and 2, Journal of Optimization Theory and Applications, Vol. 37, pp. 499–521, 1982 and Vol. 37, IV 523–543, 1982.

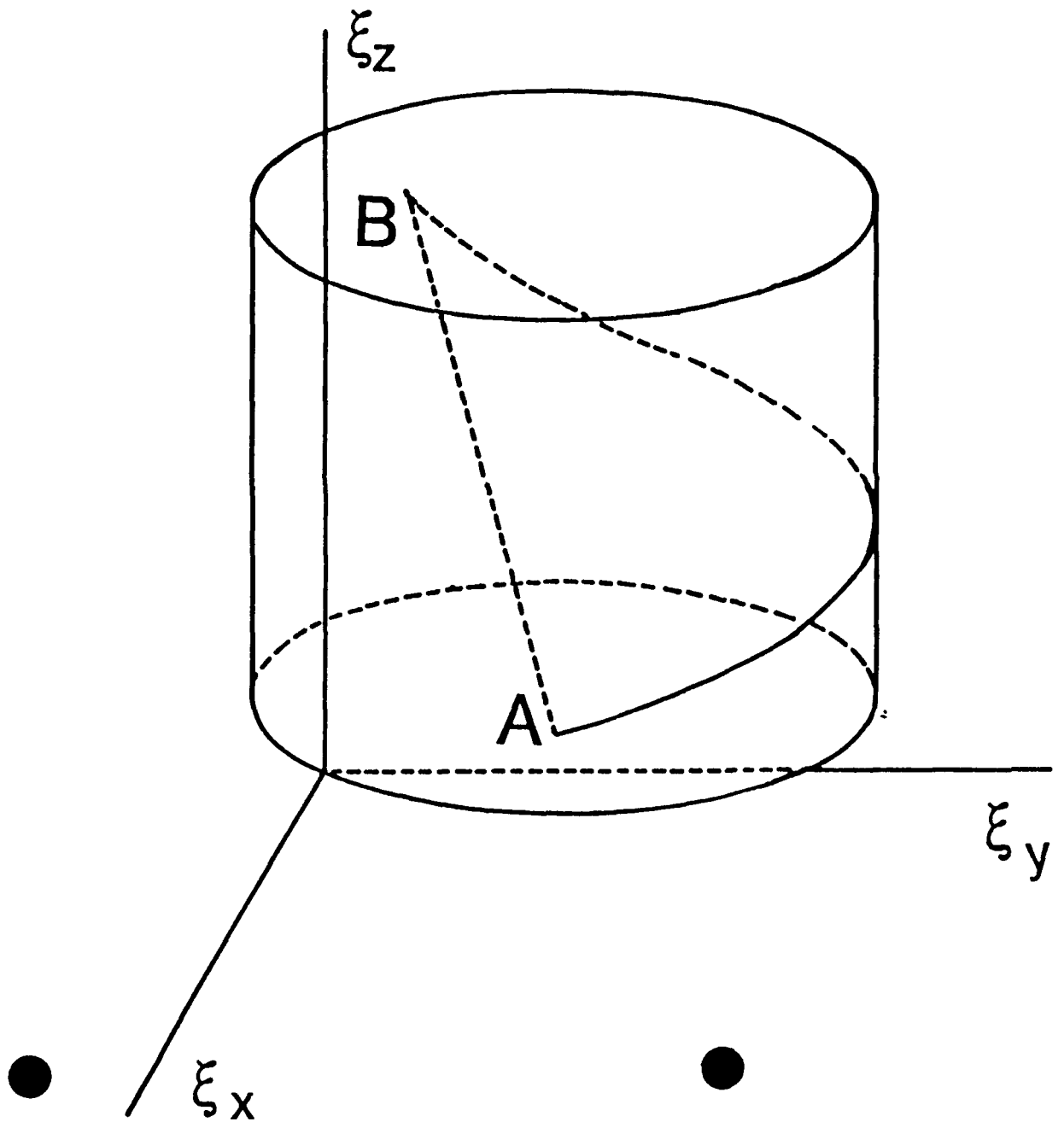
3. K.A. LURIE and A.V. CHERKAEV, G–Closure of a Set of Anisotropically Conducting Media in the Two–Dimensional Case, Journal of Optimization Theory and Applications, Vol. 42, pp. 283–304, 1984.

4. J.M. BALL, Convexity Conditions and Existence Theorems in Nonlinear Elasticity, Archive for Rational Mechanics and Analysis, Vol. 63, No. 4, pp. 337–403, 1977.

5. G. STRANG, The Polyconvexification of F(∇u), Australian National University, Research Report CMA–R09–83, 1983.

6. R.V. KOHN and G. STRANG, Optimal Design and Relaxation of Variational Problems, Parts I, II and III, Communications on Pure and Applied Mathematics, Vol. 39, No. 1, pp. 113–137, 1986; Vol. 39, No. 2, pp. 139–182, 1986, and Vol. 39, No. 3, pp. 353–377, 1986.

7. A.I. LURIE, Nonlinear Theory of Elasticity, North–Holland, Amsterdam, New York, Oxford, Tokyo, 1990. (Translation of the Russian edition, Nauka, Moscow, 1980.)


ξ_z
B
A
ξ_y
ξ_x

11. JONGEN, H. T., JONKER, P., and TWILT, F., *Nonlinear Optimization in $\mathbb{R}^n$, Vol. 2: Transversality, Flows, Parametric Aspects*, Peter Lang Verlag, Frankfurt am Main, Germany, 1986.
12. FIACCO, A. V., *Introduction to Sensitivity and Stability Analysis in Nonlinear Programming*, Academic Press, New York, New York, 1983.
13. LJUSTERNIK, L. A., and SOBOLEW, W. I., *Elemente der Funktionalanalysis*, Akademie-Verlag, Berlin, Germany, 1960.
14. CHENEY, E. W., *Introduction to Approximation Theory*, McGraw-Hill, New York, New York, 1966.
15. CLARKE, F. H., *Optimization and Nonsmooth Analysis*, John Wiley and Sons, New York, New York, 1983.
16. JONGEN, H. T., and PALLASCHKE, D., *On Linearization and Continuous Selections of Functions*, Optimization, Vol. 19, pp. 343-353, 1988.
17. AMANN, H., *Gewöhnliche Differentialgleichungen*, Walter de Gruyter, Berlin, Germany, 1983.
18. BROECKER, T., and LANDER, L., *Differentiable Germs and Catastrophes*, Cambridge University Press, Cambridge, England, 1975.

JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS: Vol. 72, No. 3, MARCH 1992

Direct Solution of an Optimal Layout Problem for Isotropic and Anisotropic Heat Conductors on a Plane[1]

K. A. LURIE[2]

Abstract. This paper suggests a procedure of direct construction of the minimal extension of a constrained optimization problem for a heat equation containing controls in the coefficients. For a somewhat simpler problem, this approach has been initiated in Ref. 1.

Key Words. Relaxation techniques, nonsaddle integrands, quasisaddle functions.

1. Introduction

We consider non-self-adjoint optimization problems for a system of equations on a plane,

$$q = \mathcal{D} \cdot \nabla T, \qquad \nabla \cdot q = 0, \tag{1}$$

where $T = T(x, y)$ denotes the temperature, and the tensor $\mathcal{D} = \mathcal{D}(x, y)$ of heat conductance plays the role of control. The set U of admissible values of $\mathcal{D}$ includes two elements (materials),

$$U = \{\mathcal{D}: \mathcal{D}_1 = uE,\ E = e_1e_1 + e_2e_2,\ \mathcal{D}_2 = \mathcal{D}_2(x, y) = R\mathcal{D}_{20}R^T,\ R = e_1'e_1 + e_2'e_2,$$
$$R^T = e_1e_1' + e_2e_2',\ \mathcal{D}_{20} = d_1e_1e_1 + d_2e_2e_2,\ 0 < u < d_1 < d_2\}, \tag{2}$$

where the unit vectors $e_1' = e_1'(x, y)$, $e_2' = e_2'(x, y)$ form an orthogonal pair which can be arbitrarily oriented at any point (x, y) of the planar domain S relative to some fixed basic pair e_1, e_2.

It is required to find the distribution

$$\mathcal{D}(x, y) = \chi_1(x, y)\mathcal{D}_1 + \chi_2(x, y)\mathcal{D}_2(x, y)$$

[1]This research has been supported by AFOSR Grant No. 90-0268. The author acknowledges discussions with Dr. Janos Turi.

[2]Professor, Department of Mathematical Sciences, Worcester Polytechnic Institute, Worcester, Massachusetts.

value problem obtained when Eqs. (1) are complemented by the linear boundary condition

$$L(T) = f \tag{3}$$

along ∂S. To specify the problem, we will consider the Dirichlet condition (s denotes the arc length of ∂S),

$$T|_{\partial S} = f(s), \tag{4}$$

and the functional

$$I(T) = -\int_S [T(x, y) - T_0(x, y)]^2 \, dx \, dy, \tag{5}$$

where $T_0(x, y) \in L_2(S)$.

This problem is known to be ill-posed and therefore requiring relaxation, i.e., the construction of an appropriate minimal extension of the initial set U of admissible controls. Such an extension can be constructed on the basis of a precise knowledge of the G-closure of U, i.e., the set GU of invariants of the effective heat conductance tensors $\mathcal{D}_0$ of all composites assembled from the elements of U (Ref. 2). However, the G-closures are known only for a few particular examples, the case of U defined by (2) among them (Ref. 3). Yet, for these selected examples, the construction of GU represents a difficult problem. For more complex situations (e.g., that of an elliptic equation of the fourth order), the problem of constructing the G-closure still remains open.

At the same time, for many applications, we do not need to know the GU set in full detail. Instead, it is often enough to specify some linear combinations of components of $\mathcal{D}_0$, particularly, for our example the combination $\mathcal{D}_0 \cdot \nabla T$ entering the first of equations (1). This is the only combination which really matters for our purposes; to determine it, we apply a direct approach, free from any reference to the G-closure.

A similar problem for $d_1 = d_2$ has been discussed in Ref. 1.

2. Transformation of the Problem

We first reduce the problem to the convenient sup inf form. Introduce the Lagrange multiplier λ and consider the augmented functional

$$J = J(T, \lambda) = I(T) - \int_S \lambda \nabla \cdot \mathcal{D} \cdot \nabla T \, dx \, dy, \tag{6}$$

following from (1).

Equating to zero the first variation of (6) with respect to T and bearing (4) in mind, we arrive at the conjugate equation

$$\nabla \cdot \mathcal{D} \cdot \nabla \lambda = -2(T - T_0) \tag{8}$$

and the boundary condition

$$\lambda|_{\partial S} = 0. \tag{9}$$

After integration by parts with the boundary condition (9), the functional (6) takes on the form

$$J = I(T) + \int_S \nabla \lambda \cdot \mathcal{D} \cdot \nabla T \, dx \, dy, \tag{10}$$

which is convenient for subsequent use.

It can be shown that the problem

$$\sup_{\mathcal{D}} I, \qquad \text{s.t. (1) and (4)},$$

is equivalent to the problem

$$\sup_{\mathcal{D}, T} \inf_{\lambda} J, \qquad \text{s.t. (4) and (9)}.$$

Indeed, since

$$J = I(T) + \int_{\partial S} \lambda n \cdot \mathcal{D} \cdot \nabla T \, ds - \int_S \lambda \nabla \cdot \mathcal{D} \cdot \nabla T \, dx \, dy,$$

the operation $\inf_\lambda J$ yields, in view of (9),

$$\inf_{\lambda} J = I(T),$$

the constraint (7) now appearing as a necessary condition for a minimum in λ.

The functional $\sup_{\mathcal{D}, T} \inf_\lambda J$ has the following upper bound:

$$\sup_{\mathcal{D}, T} \inf_{\lambda} J = \sup_{T} \sup_{\mathcal{D}} \inf_{\lambda} J \le \sup_{T} \inf_{\lambda} \sup_{\mathcal{D}} J$$

$$= \sup_{T} \inf_{\lambda} \left[-\int_S (T - T_0)^2 \, dx \, dy + \int_S G(\nabla T, \nabla \lambda) \, dx \, dy \right], \tag{11}$$

where

$$G(\xi, \eta) = \begin{cases} [(d_2 - d_1)/2] |\xi| |\eta| + [(d_2 + d_1)/2] \xi \cdot \eta, & \text{if } \Phi(\xi, \eta) \ge 0, \\ u\xi \cdot \eta, & \text{if } \Phi(\xi, \eta) \le 0. \end{cases} \tag{12}$$

Here,

$$\xi = \nabla T, \qquad \eta = \nabla \lambda, \tag{13a}$$

$$\Phi = \Phi(\xi, \eta) = \alpha|\xi| \cdot |\eta| + \beta \xi \cdot \eta, \tag{13b}$$

$$\alpha = [(d_2 - d_1)/2] > 0, \qquad \beta = [(d_2 + d_1)/2] - u > 0. \tag{13c}$$

The function $G(\xi, \eta)$ is convex with respect to any of its arguments, but nonconvex with respect to their union. The problem

$$\sup_{T} \inf_{\lambda} \left[-\int_S (T - T_0)^2 \, dx \, dy + \int_S G(\nabla T, \nabla \lambda) \, dx \, dy \right], \qquad \text{s.t. } T \in (4), \ \lambda \in (9),$$

is still ill-posed. It would be well-posed if the integrand $G(\xi, \eta)$ were a saddle function, i.e., concave in ξ for fixed η and convex in η for fixed ξ. The solution would then exist and the operations sup and inf would commute. For our problem, it is obviously not the case. However, the requirement that the function $G(\xi, \eta)$ be saddle is too restrictive when ξ and η are gradients; to ensure the existence of sup inf for this case, it is enough to require that this function be only quasisaddle (Ref. 1). We will construct its quasisaddle envelope $G^{**}(\xi, \eta)$ applying the s.c. polysaddlification transformation introduced in Ref. 1 and playing the same role in sup inf problems as the polyconvexification transformation (Refs. 4-6) plays for the infimum problems. This new transformation is given by the formula

$$G^{**}(\xi, \eta) = \sup_{A} \sup_{b} \inf_{a} \left\{ a \cdot \xi + b \cdot \eta + A(\xi_1 \eta_2 - \xi_2 \eta_1) - \inf_{\xi} \sup_{\eta} [a \cdot \xi + b \cdot \eta + A(\xi_1 \eta_2 - \xi_2 \eta_1) - G(\xi, \eta)] \right\}; \tag{14}$$

if $G(\xi, \eta)$ is convex in η and arbitrary in ξ, then (Ref. 1)

$$G^{**}(\xi, \eta) \geq G(\xi, \eta). \tag{15}$$

3. Computation of the Transformation $G^{**}(\xi, \eta)$: Upper Bound for the Functional $\sup_{\mathcal{T}, T} \inf_{\lambda} J$

We first compute

$$h(\xi, b) = \sup_{\eta} [b \cdot \eta - H(\xi, \eta)],$$

with

$$H(\xi, \eta) = -A(\xi_1 \eta_2 - \xi_2 \eta_1) + G(\xi, \eta).$$

Specifically,

$$b \cdot \eta - H(\xi, \eta) = \begin{cases} c \cdot \eta, & \text{if } \eta \in \{\Phi(\xi, \eta) \leq 0\}, \\ c \cdot \eta - \Phi, & \text{if } \eta \in \{\Phi(\xi, \eta) \geq 0\}. \end{cases}$$

The vector c has the following components:

$$c = (b_1 - A\xi_2 - u\xi_1, \, b_2 + A\xi_1 - u\xi_2). \tag{16}$$

Note also that $\Phi(\xi, \eta) \leq 0$ necessarily implies $\xi \cdot \eta \leq 0$. Desiring to compute $\sup_{\eta}[b \cdot \eta - H(\xi, \eta)]$, consider first the case $\eta \in \{\Phi(\xi, \eta) \leq 0\}$; see Fig. 1. It is obvious that

$$\sup_{\eta \in \{\Phi \leq 0\}} c \cdot \eta = \begin{cases} +\infty, & \text{if } c \notin \bar{K}, \\ 0, & \text{if } c \in \bar{K}, \end{cases} \tag{17}$$

where $\bar{K}$ is a closed sector of angular span 2ω bisected by the vector ξ; the value of ω is defined by

$$\cos \omega = 2\sqrt{(\bar{d}_1 \bar{d}_2)}/(\bar{d}_1 + \bar{d}_2), \tag{18}$$

where

$$\bar{d}_1 = d_1 - u, \qquad \bar{d}_2 = d_2 - u. \tag{19}$$

Note that the sides of $\bar{K}$ are perpendicular to the rays η_- and η_+ along which $\Phi = 0$. Disregarding the case

$$\sup_{\eta \in \{\Phi \leq 0\}} c \cdot \eta = +\infty,$$

we may assume that $c \in \bar{K}$.

Bearing this in mind, consider now the case

$$\Phi(\xi, \eta) \geq 0;$$

we have to compute

$$\sup_{\eta \in \{\Phi \geq 0\}} (c \cdot \eta - \Phi) = \sup_{\eta \in \{\Phi \geq 0\}} [(c - \beta \xi) \cdot \eta - \alpha|\xi||\eta|].$$

Assume first that

$$c - \beta \xi \in \{\Phi \leq 0\};$$

the $\sup_{\eta \in \{\Phi \geq 0\}}$ is then achieved for η parallel either to η_- or to η_+ (Fig. 1). For each of the two possibilities, we have $\Phi = 0$, which means that

$$\sup_{\eta \in \{\Phi \geq 0\}} = \max(c \cdot \eta_-, c \cdot \eta_+);$$

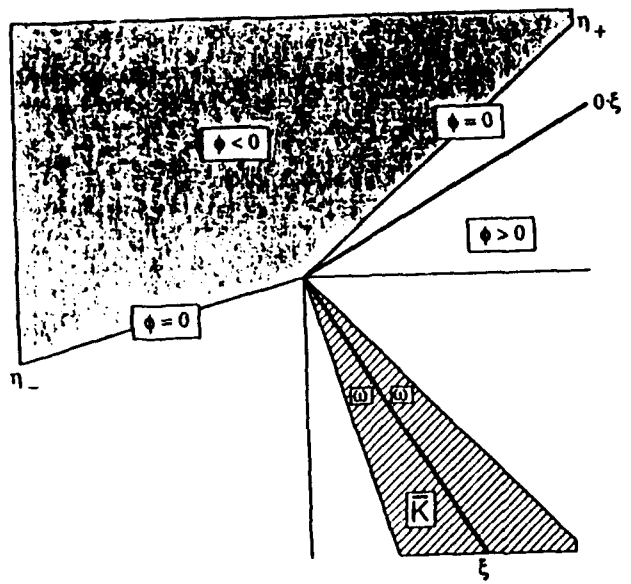


Fig. 1. Domains $\Phi > 0$ and $\Phi < 0$ in the plane of the vectors ξ, η.

but for $c \in \bar{K}$, each of $c \cdot \eta_+$ and $c \cdot \eta_-$ is either negative or zero. We thus conclude that, for $c - \beta\xi \in \{\Phi \le 0\}$,

$$\sup_{\eta \in \{\Phi \ge 0\}} (c \cdot \eta - \Phi) = 0.$$

It is now apparent that the case $c - \beta\xi \in \{\Phi \le 0\}$ does not give rise to any additional restrictions on vectors c such that

$$\sup_{\eta} [b \cdot \eta - H(\xi, \eta)] = 0;$$

the inclusion $c \in \bar{K}$ still remains the only acting requirement.

Consider now the case $c - \beta\xi \in \{\Phi \ge 0\}$. Because $\eta \in \{\Phi \ge 0\}$, the supremum

$$\sup_{\eta \in \{\Phi \ge 0\}} (c \cdot \eta - \Phi) = \sup_{\eta \in \{\Phi \ge 0\}} [(c - \beta\xi)\eta - \alpha|\xi||\eta|]$$

will be achieved for η parallel to $c - \beta\xi$, and it will be equal to

$$\sup_{|\eta|} [|c - \beta\xi| - \alpha|\xi|]|\eta|.$$

The latter value is zero provided that

$$|c - \beta\xi| - \alpha|\xi| \le 0 \tag{20}$$

and $+\infty$ otherwise. We conclude that

$$\sup_{\substack{\eta \in \{\Phi \ge 0\} \\ c \in \bar{K}}} (c \cdot \eta - \Phi) = \begin{cases} 0, & \text{if } c - \beta\xi \in \{\Phi \le 0\}, \text{ or} \\ & \text{if } c - \beta\xi \in \{\Phi \ge 0\} \text{ and } |c - \beta\xi| - \alpha|\xi| \le 0, \\ +\infty, & \text{otherwise.} \end{cases} \tag{21}$$

The vector c can be presented as

$$c = c_1\xi + c_2\mathcal{O} \cdot \xi, \qquad \mathcal{O} = -ij + ji, \tag{22a}$$

$$c_1 = (c \cdot \xi)/|\xi|^2, \qquad c_2 = (c \cdot \mathcal{O} \cdot \xi)/|\xi|^2; \tag{22b}$$

because $c \in \bar{K}$, we obviously have $c_1 > 0$; see Fig. 1. The restriction $c - \beta\xi \in \{\Phi \ge 0\}$ holds if $c_1 - \beta \ge 0$; it also holds if $c_1 - \beta \le 0$ and

$$(\beta - c_1)/|c_2| \le \tan\omega = [(\bar{d}_2 - \bar{d}_1)/2]\sqrt{(\bar{d}_2\bar{d}_1)} = k. \tag{23}$$

These inequalities, combined with the inclusion $c \in \bar{K}$, outline the part of $\bar{K}$ bounded by the rays NM, QR and the segments PN, PQ which intersect the rays at the right angles (Fig. 2). Along PN and PQ, Ineq. (23) is satisfied

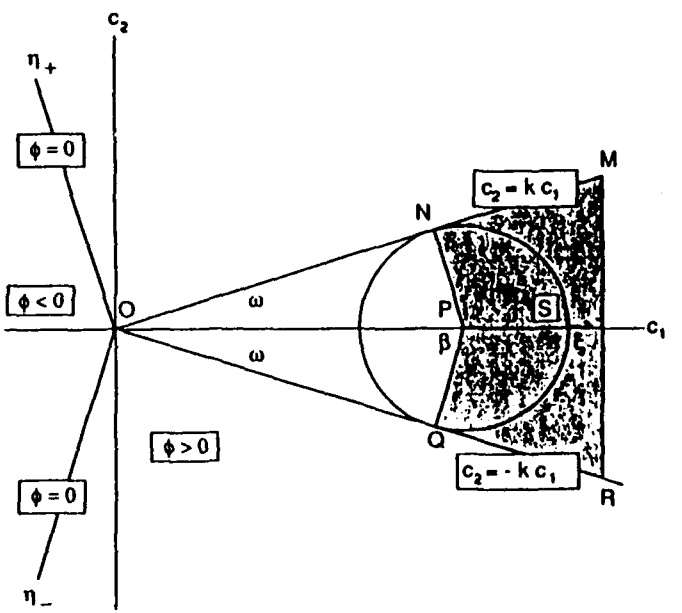


Fig. 2. Domains $\Phi > 0$ and $\Phi < 0$ in the (c_1, c_2)-pl

as strict equality; in other words, $\Phi = 0$ along these segments. The mentioned part of $\bar{K}$ is shaded in Fig. 2.

To characterize the domain for which

$$\sup_{\substack{\eta \in \{\Phi \geq 0\} \\ c \in \bar{K}, c - \beta\xi \in \{\Phi \geq 0\}}} (c \cdot \eta - \Phi) = 0, \tag{24a}$$

we should also consider the restriction

$$|c - \beta\xi| - \alpha|\xi| \leq 0.$$

This inequality holds for the interior of a circle

$$(c_1 - \beta)^2 + c_2^2 = \alpha^2, \tag{24b}$$

with center at P and radius $|PN|$ equal to α. The intersection of (24b) and the domain shaded in Fig. 2 (i.e., the circular sector NSQPN) represents the region for which (24a) holds. If we now combine this sector with the domain ONPQ where the inclusions $c \in \bar{K}$, $c - \beta\xi \in \{\Phi \leq 0\}$ take place, then we will arrive at the complete geometric description of a set where

$$\bar{h}(\xi, b) = \sup_{\eta}[b \cdot \eta - H(\xi, \eta)] = 0.$$

This set is represented by the figure ONSQO on the (c_1, c_2)-plane (Fig. 2). For points outside this figure,

$$\bar{h}(\xi, b) = \begin{cases} 0, & \xi \in \text{ONSQO}, \\ +\infty, & \xi \notin \text{ONSQO}. \end{cases} \tag{25}$$

To characterize the domain ONSQO in terms of ξ, one has to apply Eqs. (16) and (22). The circle (24b) is mapped onto the circle

$$(A^2 + d_1 d_2)\xi^2 - (d_1 + d_2)b \cdot \xi + 2A(\xi_1 b_2 - \xi_2 b_1) + b^2 = 0, \tag{26a}$$

$$\xi^2 = \xi_1^2 + \xi_2^2, \qquad b^2 = b_1^2 + b_2^2, \tag{26b}$$

and the straight segments ON, OQ become circular segments in the ξ-plane,

$$\text{ON:} \quad (A + ku)\xi^2 - kb \cdot \xi + (\xi_1 b_2 - \xi_2 b_1) = 0, \tag{27}$$

$$\text{OQ:} \quad (A - ku)\xi^2 + kb \cdot \xi + (\xi_1 b_2 - \xi_2 b_1) = 0. \tag{28}$$

The ξ-image of ONSQO is illustrated in Fig. 3a for $|A| < ku$ and in Fig. 3b for $|A| > ku$. The domain ONSQO is convex in the first case and nonconvex in the second case. When $|A| = ku$, one of the circles (27), (28) becomes a straight line.

In accordance with (14), the operation

$$\inf_{a}\left\{a \cdot \xi - \inf_{\xi}[a \cdot \xi - (-\bar{h}(\xi, b))]\right\} \tag{29}$$

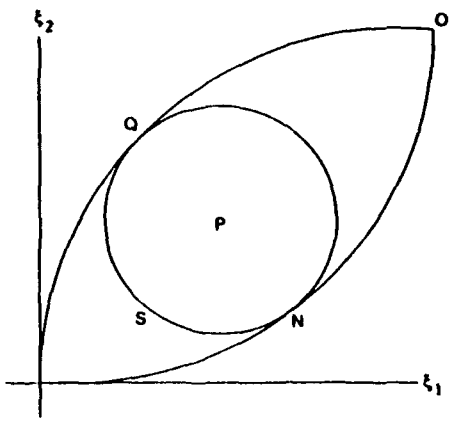


Fig. 3a. Region OQSNO in the ξ-plane for $|A| < ku$.

is now to be executed. This operation constructs the concave ξ-envelope of $-\bar{h}(\xi, b)$; if this function is itself concave, then it is simply recovered by (29). In our case, the function $-\bar{h}(\xi, b)$ is defined as [Eq. (25)]

$$-\bar{h}(\xi, b) = \begin{cases} 0, & \xi \in \text{ONSQO}, \\ +\infty, & \xi \notin \text{ONSQO}. \end{cases}$$

This function is concave provided that the figure ONSQO is convex. We already noticed that this condition is satisfied for $|A| < ku$ (Fig. 3a), but is

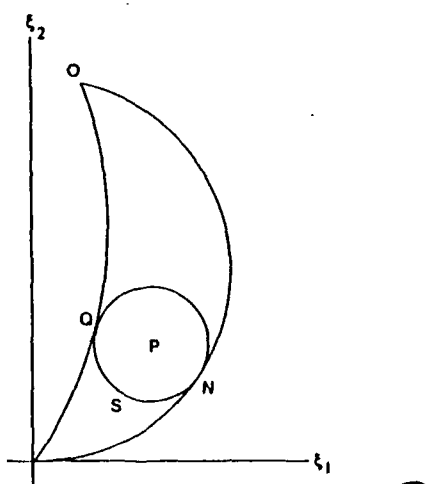


Fig. 3b. Region OQSNO in the ξ-plane for $|A| >$

violated for $|A| > ku$ (Fig. 3b). The latter case therefore requires convexification of the domain ONSQO. The procedure is illustrated by Fig. 4.

Any point ξ of the circle (26a) can be represented in a parametric form by the formulas

$$\xi_1 = \{[(d_1 + d_2)/2]b_1 - Ab_2 + [(d_2 - d_1)/2]|b| \cos \varphi\}/(A^2 + d_1 d_2), \tag{30a}$$

$$\xi_2 = \{[(d_1 + d_2)/2]b_2 + Ab_1 + [(d_2 - d_1)/2]|b| \sin \varphi\}/(A^2 + d_1 d_2); \tag{30b}$$

here, φ is a parameter. Convexification of ONSQO requires the construction of a tangent OT, the point T also determined by Eqs. (30) for some selected value of φ. To find this value, we use the relationships

$$(\xi - \xi_O) \cdot (\xi_T - \xi_P) = 0, \tag{31}$$

$$(\xi_T - \xi_O) \cdot (\xi_T - \xi_P) = 0. \tag{32}$$

Here, ξ is any point of the tangent, ξ_O denotes the ξ-vector at O, and ξ_T, ξ_P denote the ξ-vectors at T and P.

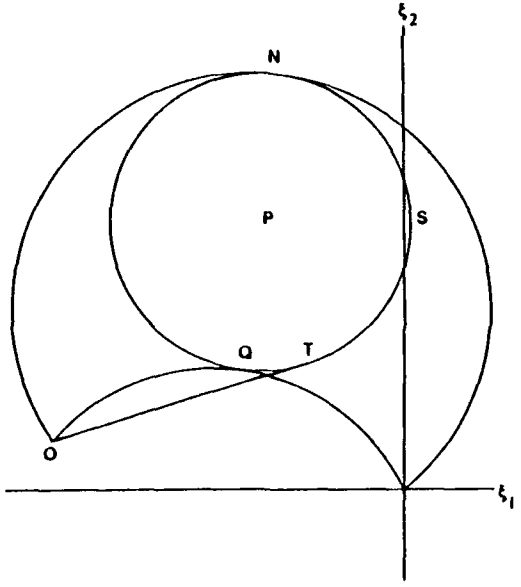


Fig. 4. Convex envelope OTSNO of the nonconvex region OQSNO in the ξ-plane for $|A| > ku$.

The following formulas for ξ_O and ξ_P are obvious:

$$\xi_O = i[(ub_1 - Ab_2)/(A^2 + u^2)] + j[(Ab_1 + ub_2)/(A^2 + u^2)], \tag{33}$$

$$\xi_P = i\{[(d_1 + d_2)/2]b_1 - Ab_2\}/(A^2 + d_1 d_2) + j\{[(d_1 + d_2)/2]b_2 + Ab_1\}/(A^2 + d_1 d_2). \tag{34}$$

The vector ξ_O satisfies both Eqs. (27) and (28), and the expression (34) follows from (26a).

Equations (30) for ξ_T show that

$$\xi_T = \xi_C + (i \cos \varphi + j \sin \varphi)[(d_2 - d_1)/2]|b|/(A^2 + d_1 d_2).$$

Equations (31), (32) now take on the form

$$(A^2 + u^2)(\xi_1 \cos \varphi + \xi_2 \sin \varphi) = u(b_1 \cos \varphi + b_2 \sin \varphi) + A(b_1 \sin \varphi - b_2 \cos \varphi), \tag{35}$$

$$\{[(d_1 + d_2)/2(A^2 + d_1 d_2)] - [u/(A^2 + u^2)]\}(b_1 \cos \varphi + b_2 \sin \varphi) + A\{[1/(A^2 + d_1 d_2)] + [1/(A^2 + u^2)]\}(b_1 \sin \varphi - b_2 \cos \varphi) = -[(d_2 - d_1)/2(A^2 + d_1 d_2)]|b|. \tag{36}$$

Introduce now the angles θ, χ by the formulas (Fig. 5)

$$b_1 = |b| \cos \theta, \qquad b_2 = |b| \sin \theta, \qquad \chi = \varphi - \theta, \tag{37}$$

and compute the combinations

$$b_1 \cos \varphi + b_2 \sin \varphi = |b| \cos \chi,$$

$$b_1 \sin \varphi - b_2 \cos \varphi = |b| \sin \chi,$$

$$\xi_1 \cos \varphi + \xi_2 \sin \varphi = \xi_1(\cos \chi \cos \theta - \sin \chi \sin \theta) + \xi_2(\sin \chi \cos \theta + \cos \chi \sin \theta) = [(\xi \cdot b)/|b|] \cos \chi - [(\xi_1 b_2 - \xi_2 b_1)/|b|] \sin \chi.$$

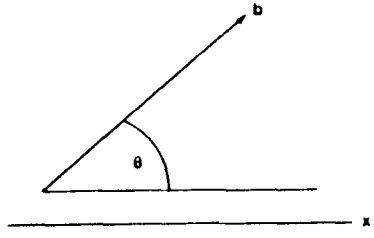


Fig. 5. Vector b.

Equations (35) and (36) are now rewritten as

$$(u + A\tan\chi)b^2 - (A^2 + u^2)D \cdot b = 0, \tag{38}$$

$$M\cos\chi + N\sin\chi = -P, \tag{39}$$

where D is a vector with two components,

$$D = (\xi_1 + \xi_2\tan\chi,\ \xi_2 - \xi_1\tan\chi), \tag{40}$$

and M, N, P are scalar,

$$M = A^2\{[(d_1 + d_2)/2] - u\} + u\{[u(d_1 + d_2)/2] - d_1d_2\}, \tag{41a}$$

$$N = -A(d_1d_2 - u^2), \tag{41b}$$

$$P = (d_2 - d_1)(A^2 + u^2)/2. \tag{41c}$$

The angle χ defined by (39) depends on A, u, d_1, d_2; its cosine is computed to be equal to

$$(\cos\chi)_{1,2} = [-MP \pm N\sqrt{(N^2 + M^2 - P^2)}]/(M^2 + N^2). \tag{42}$$

It is easy to show that [see (19) and (41)]

$$N^2 + M^2 - P^2 = \bar{d}_1\bar{d}_2(A^2 + u^2)(A^2 + d_1d_2).$$

Note also that

$$(\sin\chi)_{1,2} = [-PN \mp M\sqrt{(N^2 + M^2 - P^2)}]/(M^2 + N^2), \tag{43}$$

$$(\tan\chi)_{1,2} = [MN \pm P\sqrt{(N^2 + M^2 - P^2)}]/(P^2 - N^2). \tag{44}$$

Returning to the convexification procedure, we conclude that the operation (29) recovers $-\bar{h}(\xi, b)$ if $|A| < ku$, and gives birth to a new function defined as (see Fig. 4)

$$-\bar{h}(\xi, b) = \begin{cases} 0, & \xi \in \text{OTSNO}, \\ -\infty, & \text{otherwise}, \end{cases} \tag{45}$$

for $|A| \geq ku$.

To go on with the transformation (14), one has first to describe the domain of zero value of the function (29) treated as a function of b for fixed ξ. The domain OQSNO on the b-plane obviously has a shape of a sector bounded by two radii ON and OQ [the curves (27) and (28) perceived in terms of b] and the part QSN of a circle which is given by Eq. (26a); see Fig. 6a. This sector is a required domain for $|A| \leq ku$. As to the case $|A| \geq ku$, the corresponding domain should be the b-interpretation of a convexified region OTSNO on a ξ-plane. On the b-plane, the domain OTSNO is characterized by the appearance of a circular arc OT instead of

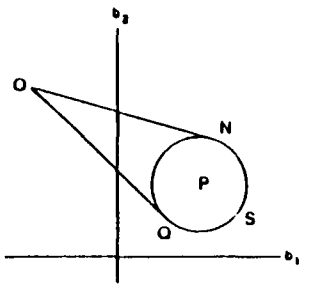


Fig. 6a. Region OQSNO in the b-plane for $|A| > ku$.

a straight line OQ as a part of the boundary. The curve OT (see Fig. 6b) is the b-interpretation of a straight line OT which has appeared in the course of convexification procedure held on a ξ-plane (Fig. 4). The straight line OQ on the b-plane (Fig. 6b) corresponds to a circular arc OQ on the ξ-plane (Figs. 3b and 4), this latter arc located within the convex hull OTSNO of the original domain OQSNO. It is no surprise that the b-image of OQ (a straight line on Fig. 6b) also belongs to the interior of OTSNO.

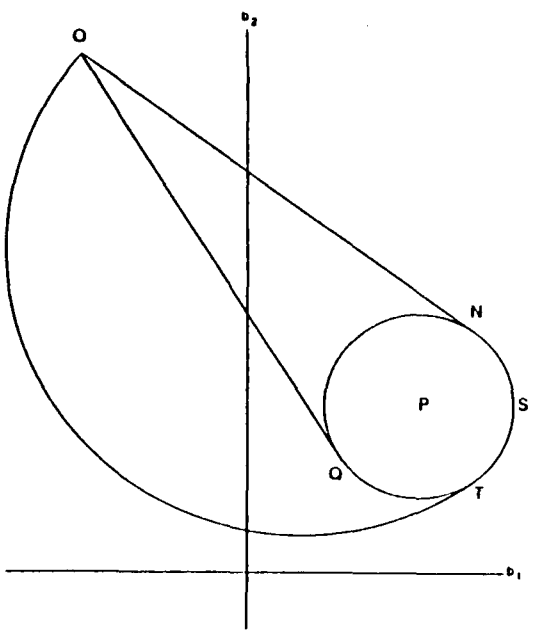


Fig. 6b. Regions OQSNO and OTSNO in the b-plane for $|A| > ku$.

The circular arc OT on the b-plane is defined by Eq. (38). We have finally the following results: for $|A| \le ku$,

$$\inf_a \left\{ a \cdot \xi - \inf_\xi [a \cdot \xi - (-\bar{h}(\xi, b))] \right\} = \begin{cases} 0, & \xi \in \text{OQSNO}, \\ -\infty, & \xi \notin \text{OQSNO}; \end{cases} \quad (46)$$

for $|A| \ge ku$,

$$\inf_a \left\{ a \cdot \xi - \inf [a \cdot \xi - (\bar{h}(\xi, b))] \right\} = \begin{cases} 0, & \xi \in \text{OTSNO}, \\ -\infty, & \xi \notin \text{OTSNO}. \end{cases} \quad (47)$$

The next step in (14),

$$\sup_b \left(b \cdot \eta + \inf_a \left\{ a \cdot \xi - \inf_\xi [a \cdot \xi - (-\bar{h}(\xi, b))] \right\} \right),$$

is thus reduced to

$$\sup_{b \in \text{OQSNO}} b \cdot \eta, \qquad \text{if } |A| \le ku, \quad (48)$$

$$\sup_{b \in \text{OTSNO}} b \cdot \eta, \qquad \text{if } |A| \ge ku. \quad (49)$$

The final choice between these two possibilities is decided by the closing operation

$$\sup_A \left[A(\xi_1 \eta_2 - \xi_2 \eta_1) + \begin{cases} \sup_{b \in \text{OQSNO}} b \cdot \eta, & \text{if } |A| \le ku \\ \sup_{b \in \text{OTSNO}} b \cdot \eta, & \text{if } |A| \ge ku \end{cases} \right]. \quad (50)$$

The first part of this operation ($|A| \le ku$) is easy to compute. Due to the convexity of the domain OQSNO, the point b^* which realizes $\sup b \cdot \eta$ belongs to the boundary of this domain; namely, it either is the vertex O or it belongs to the circular arc QSN (Fig. 6a). In the latter case, the point b^* should be a stationary point of the function [see (26a)]

$$g(b) = b \cdot \eta + \mu[b^2 + (A^2 + d_1 d_2)\xi^2 + 2A(\xi_1 b_2 - \xi_2 b_1) - (d_1 + d_2)\xi \cdot b],$$

where μ is a Lagrange multiplier. An elementary calculation shows that

$$g(b^*) = -A(\xi_1 \eta_2 - \xi_2 \eta_1) + [(d_2 - d_1)/2]|\xi||\eta| + [(d_1 + d_2)/2]\xi \cdot \eta,$$

and consequently,

$$A(\xi_1 \eta_2 - \xi_2 \eta_1) + g(b^*) = [(d_2 - d_1)/2]|\xi||\eta| + [(d_1 + d_2)/2]\xi \cdot \eta.$$

The latter expression does not depend on A and obviously recovers the original function $G(\xi, \eta)$; see (12), first line. This regime holds as long as the vector η stays out of the shaded sector bounded by rays perpendicular to the sides of the angle QON, in other words, while η belongs to the range $\Phi \ge 0$ (see Fig. 1). If η violates this inequality, then the vertex O becomes the point b^*, and we completely recover $G(\xi, \eta)$ as given by Eq. (12).

The $\sup_A$ in (46) thus does not depend on A for $|A| \le ku$; it is given by $G(\xi, \eta)$ in (12). As to the case $|A| \ge ku$, the situation is different; we shall see that the global $\sup_A$ in (46) is utterly determined by this case.

The point b^* at which $\sup_{b \in \text{OTSNQ}} b \cdot \eta$ is now attained belongs to the boundary of a convex domain OTSNO. Now, however, we meet three possibilities instead of the two that we had before: the point b^* may (i) belong to the arc OT, (ii) lie on the arc TSN [see (26)], or (iii) coincide with O. The first possibility requires finding the stationary point of the function

$$g = b \cdot \eta + \nu[(u + A \tan \chi)b^2 - (A^2 + u^2)D \cdot b],$$

where ν is a Lagrange multiplier associated with the constraint (38).

It is easy to show that

$$g(b^*) = (A^2 + u^2)|\xi||\eta|/[2|u + A \tan \chi||\cos \chi|] + [\xi \cdot \eta - (\xi_1 \eta_2 - \xi_2 \eta_1) \tan \chi](A^2 + u^2)/2(u + A \tan \chi).$$

We further have

$$\begin{aligned} & g(b^*) + A(\xi_1 \eta_2 - \xi_2 \eta_1) \\ & = (A^2 + u^2)|\xi||\eta|/[2|u + A \tan \chi||\cos \chi|] \\ & + \xi \cdot \eta (A^2 + u^2)/2(u + A \tan \chi) \\ & + (\xi_1 \eta_2 - \xi_2 \eta_1)[A - (A^2 + u^2) \tan \chi / 2(u + A \tan \chi)]. \end{aligned} \quad (51)$$

To compute

$$\sup_{\substack{A, \\ |A| \le ku}} [g(b^*) + A(\xi_1 \eta_2 - \xi_2 \eta_1)],$$

one has to differentiate (51) in A and take notice that χ depends on A through (39). Differentiating the latter with respect to A, we get

$$d\chi/dA = (M_A \cos \chi + N_A \sin \chi + P_A)/(M \sin \chi - N \cos \chi), \quad (52)$$

where [see (41)]

$$M_A = 2A\{[(d_1 + d_2)/2] - u\}, \qquad N_A = -(d_1 d_2 - u^2), \quad (53a)$$

$$P_A = A(d_2 - d_1). \quad (53b)$$

The necessary condition for the function (51) to be stationary in A is given by (assume that $\cos\chi < 0$)

$$(d/dA)[g(b^*) + A(\xi_1\eta_2 - \xi_2\eta_1)]$$
$$= (1/2)[-(|\xi||\eta|/\cos\chi) + \xi\cdot\eta]\cdot d/dA[(A^2+u^2)/(u + A\tan\chi)]$$
$$-(1/2)|\xi||\eta|[(A^2+u^2)\sin\chi/(u + A\tan\chi)\cos^2\chi]\, d\chi/dA$$
$$+(\xi_1\eta_2 - \xi_2\eta_1)\{-(1/2)\tan\chi\cdot(d/dA)[(A^2+u^2)/(u + A\tan\chi)]$$
$$+1 - (A^2+u^2)(d\chi/dA)/2(u + A\tan\chi)\cos^2\chi\}(\xi_1\eta_2 - \xi_2\eta_1) = 0. \tag{54}$$

One can easily check that the values

$$A = \pm\sqrt{(ud_1\bar{d}_2/\bar{d}_1)} \tag{55}$$

satisfy (54), i.e., they are the required stationary values of A.[3] For the first of these values [+ sign in (55)], we have

$$A^2 + u^2 = u(d_1d_2 - u^2)/\bar{d}_1, \tag{56a}$$
$$A^2 + d_1d_2 = d_1(d_1d_2 - u^2)/\bar{d}_1, \tag{56b}$$
$$M = [1/(2\bar{d}_1)]u(d_2 - d_1)(d_1d_2 - u^2) = P, \tag{56c}$$
$$N = -[\sqrt{(ud_1\bar{d}_2/\bar{d}_1)}](d_1d_2 - u^2), \tag{56d}$$
$$(\cos\chi)_{1,2} = (-M^2 \pm N|N|)/(M^2+N^2)$$
$$= \begin{cases} -1, \\ -(M^2-N^2)/(M^2+N^2), \end{cases} \tag{57a}$$
$$(\tan\chi)_{1,2} = (MN \pm M|N|)/(M^2-N^2)$$
$$= \begin{cases} 0, \\ 2MN/(M^2-N^2). \end{cases} \tag{57b}$$

The expression

$$(d/dA)[(A^2+u^2)/(u + A\tan\chi)]$$
$$= 2A/(u + A\tan\chi) - (A^2+u^2)$$
$$\times[\tan\chi + A(d\chi/dA)/\cos^2\chi]/(u + A\tan\chi)^2$$

should be computed for $\cos\chi = -1$, $\tan\chi = 0$ ($\chi = \pi$). Bearing in mind Eqs. (52), (53), we get

$$d\chi/dA = (-M_A + P_A)/N = 2\bar{d}_1/(d_1d_2 - u^2),$$

and consequently,

$$(d/dA)[(A^2+u^2)/(u + A\tan\chi)] = 0.$$

Equation (5) is now seen to be satisfied. The case of minus sign in (55) is treated analogously. The stationary values (55) can be shown to provide

$$\sup_A [g(b^*) + A(\xi_1\eta_2 - \xi_2\eta_1)].$$

This regime holds if the vector η belongs to the range bounded by the normal n_O and n_T to a circular arc OT at the points O and T (see Fig. 6b).

The vector n of normal at any point of OT can be computed by (38),

$$n = 2(u + A\tan\chi)b - (A^2+u^2)D,$$

or in view of (55), (40), and $\tan\chi = 0$,

$$n = 2ub - u[(d_1\bar{d}_2/\bar{d}_1) + u]\xi. \tag{58}$$

At the point O, we have [cf. (22), (33)]

$$b = u\xi - A\mathcal{O}\xi = u\xi - \sqrt{(ud_1\bar{d}_2/\bar{d}_1)}\mathcal{O}\cdot\xi.$$

If we denote by $2\gamma_0$ the angle between ξ, η and require that $\eta \uparrow\uparrow n_O$, then this requirement will result in the relationship

$$\tan\gamma_0 = \sqrt{(d_1\bar{d}_2/u\bar{d}_1)}, \tag{59}$$

defining the critical angle γ_0 separating the regime when $b^* \in$ OT from that when b^* coincides with O. It remains to examine another extreme situation associated with the point T. At this point, since $\chi = \pi$, we have $\varphi = \pi + \theta$, and consequently [see Eqs. (30) and (37)],

$$\xi_1 = (d_1b_1 - Ab_2)/(A^2 + d_1d_2),$$
$$\xi_2 = (Ab_1 + d_1b_2)/(A^2 + d_1d_2),$$

or equivalently,

$$b_1 = (d_1\xi_1 + A\xi_2)(A^2 + d_1d_2)/(A^2 + d_1^2),$$
$$b_2 = (-A\xi_1 + d_1\xi_2)(A^2 + d_1d_2)/(A^2 + d_1^2).$$

Bearing this in mind and taking notice of Eq. (56), we obtain for n_1 [see (58)] the expression

$$n_T = [(d_1d_2 - u^2)/(u\bar{d}_2 + d_1\bar{d}_1)]u$$
$$\times\{[d_1 - (u\bar{d}_2/\bar{d}_1)]\xi - 2\sqrt{(ud_1\bar{d}_2/\bar{d}_1)}\mathcal{O}\cdot\xi\}.$$

[3] Obviously,

$$A^2 = ud_1\bar{d}_2/\bar{d}_1 > u^2\bar{d}_2/\bar{d}_1 \geq k^2u^2 = (d_2 - d_1)^2u^2/4\bar{d}_1\bar{d}_2,$$

so that the choice (55) is consistent with $|A| \geq ku$.

if we require that $\eta \| n_T$ and denote by $\angle\gamma_T$ the corresponding angle between ξ and η, then the relationship

$$\tan \gamma_T = \sqrt{(u\bar{d}_2/d_1\bar{d}_1)} \tag{60}$$

will satisfy the mentioned requirement and thus define the critical angle γ_T separating the regime when $b^* \in \mathrm{OT}$ from that when $b^* \in \mathrm{TSN}$ (Fig. 6b). For the latter regime, the transformation (14) will recover the function $G(\xi, \eta)$ as given by the first line in (12).

Summarizing the results of this section, we arrive at the following relations:

$$\sup_{\substack{A \\ |A| \le ku}} \left[A(\xi_1\eta_2 - \xi_2\eta_1) + \sup_{b \in \mathrm{OQSNO}} b \cdot \eta \right]$$

$$= \begin{cases} [(d_2 - d_1)/2]|\xi||\eta| + [(d_2 + d_1)/2]\xi \cdot \eta, & 0 \le \tan\gamma \le \sqrt{(\bar{d}_2/\bar{d}_1)}, \\ u\xi \cdot \eta, & \sqrt{(\bar{d}_2/\bar{d}_1)} \le \tan\gamma \le \infty, \end{cases} \tag{61}$$

$$\sup_{\substack{A \\ |A| \ge ku}} \left[A(\xi_1\eta_2 - \xi_2\eta_1) + \sup_{b \in \mathrm{OTSNO}} b \cdot \eta \right]$$

$$= \begin{cases} [(d_2 - d_1)/2]|\xi| \cdot |\eta| + [(d_2 + d_1)/2]\xi \cdot \eta, & 0 \le \tan\gamma \le \sqrt{(u\bar{d}_2/d_1\bar{d}_1)}, \\ [(d_1d_2 - u^2)/2\bar{d}_1][|\xi||\eta| + \xi \cdot \eta] + [\sqrt{(ud_1\bar{d}_2/\bar{d}_1)}](\xi_1\eta_2 - \xi_2\eta_1), & \\ \qquad\qquad \sqrt{(u\bar{d}_2/d_1\bar{d}_1)} \le \tan\gamma \le \sqrt{(d_1\bar{d}_2/u\bar{d}_1)}, & \\ u\xi \cdot \eta, \qquad \sqrt{(d_1\bar{d}_2/u\bar{d}_1)} \le \tan\gamma \le \infty. & \end{cases} \tag{62}$$

Obviously, the global $\sup_A$ [i.e., $G^{**}(\xi, \eta)$] is given by (62), since the latter function is pointwise greater than or equal to that defined by (61). This observation stays in full accordance with (15). We thus arrive at the following upper bound [see (11) and (15)]:

$$\sup_{\mathfrak{D},T} \inf_A J \le \sup_T \inf_A \times \left[-\int_S (T - T_0)^2 \, dx\, dy + \int_S G^{**}(\nabla T, \nabla\lambda) \, dx\, dy \right], \tag{63}$$

where $G^{**}(\xi, \eta)$ is defined by (62).

4. Special Microstructure: Lower Bound for the Functional $\sup_{\mathfrak{D},T} \inf_A J$

This functional can be estimated from below by its value calculated for some special microstructure properly assembled from the given constituents (2).

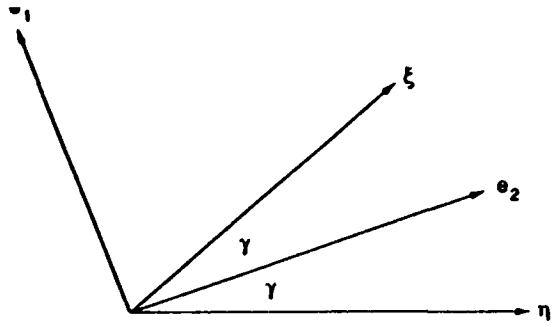


Fig. 7. Optimal orientation of the main axes e_1, e_2 with respect to the vectors ξ, η.

Consider the functional (10), and compute the integrand $\xi \cdot \mathfrak{D} \cdot \eta$ for $\mathfrak{D} = \mathfrak{D}_2$, the latter tensor having its main axes e_1, e_2 oriented as shown in Fig. 7. Obviously,

$$\begin{aligned} \xi \cdot \mathfrak{D}_2 \cdot \eta &= |\xi||\eta|(d_2 \cos^2\gamma - d_1 \sin^2\gamma) \\ &= [(d_2 - d_1)/2]|\xi||\eta| + [(d_2 + d_1)/2]|\xi||\eta| \cos 2\gamma \\ &= [(d_2 - d_1)/2]|\xi||\eta| + [(d_2 + d_1)/2]\xi \cdot \eta; \end{aligned}$$

this expressions matches the first line in (58).

Consider now the functional (10), and compute the integrand $\xi \cdot \mathfrak{D} \cdot \eta$ for $\mathfrak{D}$ defined as follows. Take the original constituents $\mathfrak{D}_1$ and $\mathfrak{D}_2$ in the relative amounts m and $1 - m$, respectively, and prepare a laminate as shown in Fig. 8. The tensor $\mathfrak{D}$ will be characterized by the relationship

$$\mathfrak{D} = \Lambda_1 e_1 e_1 + \Lambda_2 e_2 e_2, \tag{64}$$

where

$$\Lambda_1 = d_1 u/[m d_1 + (1 - m)u],$$
$$\Lambda_2 = mu + (1 - m)d_2,$$

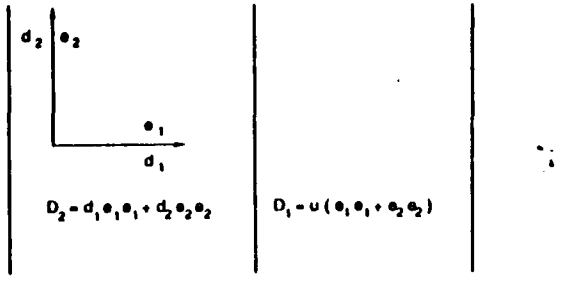


Fig. 8. Structure of optimal laminate assembled from original materials.

or after elimination of m,

$$\Lambda_1 = (ud_1\bar{d}_2/\bar{d}_1)/[u + (d_1\bar{d}_2/\bar{d}_1) - \Lambda_2]. \tag{65}$$

Compute now the integrand $\xi \cdot \mathcal{D} \cdot \eta$ for $\mathcal{D}$ defined by (64)-(65) and oriented as shown in Fig. 7. If the concentration m is chosen in accordance with the rule

$$\tan^2 \gamma = (ud_1\bar{d}_2/\bar{d}_1)/\Lambda_1^2, \tag{66}$$

then the integrand will be equal to [see (65)]

$$\begin{aligned}\xi \cdot \mathcal{D} \cdot \eta &= |\xi||\eta|(\Lambda_2 \cos^2 \gamma - \Lambda_1 \sin^2 \gamma)\\ &= |\xi||\eta| \cos^2 \gamma(\Lambda_2 - \Lambda_1 \tan^2 \gamma)\\ &= |\xi||\eta| \cos^2 \gamma[u + (d_1\bar{d}_2/\bar{d}_1) - (ud_1\bar{d}_2/\bar{d}_1)\Lambda_1 - \Lambda_1 \tan^2 \gamma];\end{aligned}$$

if we observe that

$$u + (d_1\bar{d}_2/\bar{d}_1) = [u(d_1 - u) + d_1(d_2 - u)]/\bar{d}_1 = (d_1d_2 - u^2)/\bar{d}_1$$

and make use of (62) to eliminate Λ_1, we obtain

$$\begin{aligned}\xi \cdot \mathcal{D} \cdot \eta = (d_1d_2 - u^2)|\xi||\eta|(1 + \cos 2\gamma)/(2\bar{d}_1)\\ -|\xi||\eta|[\sqrt{(ud_1\bar{d}_2/\bar{d}_1)}] \sin 2\gamma,\end{aligned}$$

which is the same as the expression in the second line of (62). Note that Eq. (66) allows one to determine $m \in [0, 1]$ provided that $u \le \Lambda_1 \le d_1$, i.e., provided

$$\sqrt{(u\bar{d}_2/d_1\bar{d}_1)} \le \tan \gamma \le \sqrt{(d_1\bar{d}_2/u\bar{d}_1)},$$

the latter inequalities reproducing the interval for $\tan \gamma$ allowed by the second line in (62).

Finally, if we compute $\xi \cdot \mathcal{D} \cdot \eta$ for $\mathcal{D} = \mathcal{D}_1$, then the result $u\xi \cdot \eta$ will fit the last line in (62).

The values of $\xi \cdot \mathcal{D} \cdot \eta$ computed above provide us with a lower bound for $\sup_{\mathcal{D},T} \inf_{\lambda} J$, because these values are associated with some special tensors $\mathcal{D}$. On the other hand, the lower bound obtained so far coincides with the upper bound rendered by the transformation (14). We therefore conclude that the problem

$$\sup_{T} \inf_{\lambda} \left[-\int_S (T - T_0)^2 \, dx \, dy + \int_S G^{**}(\nabla T, \nabla \lambda) \, dx \, dy \right]$$

under the side conditions (4), (9) is well posed. We have thus far arrived at the necessary extension of the original problem.

5. Comparison with the G-Closure Approach

The G-closure GU has been described in Ref. 3 for a set U specified by (2). This is illustrated in Fig. 9 as a shaded region bounded by the diagonal $\Lambda_1 = \Lambda_2$, the hyperbolas (65), and

$$\Lambda_1\Lambda_2 = d_1d_2. \tag{67}$$

Observe that the hyperbola (65) may be considered as initiated by two isotropic materials possessing heat conductances equal to

$$\mathcal{D}_1 = uE \quad \text{and} \quad \mathcal{D}_3 = (d_1\bar{d}_2/\bar{d}_1)E,$$

respectively. The region below (65) and above the diagonal $\Lambda_1 = \Lambda_2$ represents the G-closure, i.e., a set of all composites initiated by these materials. We make use, however, just of a part of this set, namely, the part which lies to the left of the hyperbola (67). This part, shaded in Fig. 9, is a G-closure GU of the set (2). The points of (65) are initiated as laminates assembled of $\mathcal{D}_1$ and $\mathcal{D}_2$ as illustrated in Fig. 8. The same construction could be applied to isotropic constituents

$$\mathcal{D}_1 = uE \quad \text{and} \quad \mathcal{D}_3 = (d_1\bar{d}_2/\bar{d}_1)E;$$

the only difference will be in the values of the concentration parameter m.

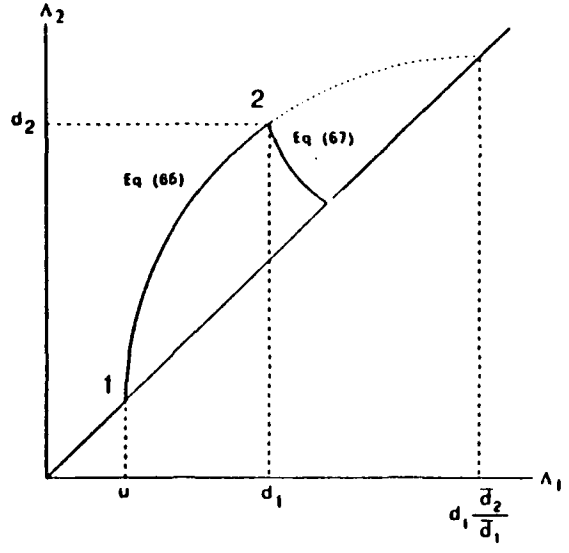


Fig. 9. G-closure initiated by the original materials.

... now apply the relaxed formulation of the original problem (1)-(3), i.e., make use of GU instead of (2), then the necessary conditions of optimality will show that only three regimes can participate in the optimal layout. These will be:

(i) the regime when $\mathcal{D} = \mathcal{D}_2$ (point 2 in Fig. 9), this tensor oriented as shown in Fig. 7 and the regime valid for $0 \le \tan\gamma \le \sqrt{(u\bar{d}_2/d_1\bar{d}_1)}$;

(ii) the regime when $\mathcal{D}$ corresponds to some point belonging to the boundary 12 in Fig. 9, the tensor $\mathcal{D}$ oriented as shown in Fig. 7 and its eigenvalue Λ_1 chosen in accordance with (62); this regime is valid for $\sqrt{(u\bar{d}_2/d_1\bar{d}_1)} \le \tan\gamma \le \sqrt{(d_1\bar{d}_2/u\bar{d}_1)}$;

(iii) the regime $\mathcal{D} = \mathcal{D}_1$ (point 1 in Fig. 9), valid for $\sqrt{(d_1\bar{d}_2/u\bar{d}_1)} \le \tan\gamma \le \infty$.

We thus obtain the same results as given by Eq. (62).

6. Conclusions

The method of direct relaxation of ill-posed problems of optimal design is applied to the case where the original materials are isotropic and anisotropic. The relaxed formulation appears as a result of elementary calculations which can be easily generalized to more complex situations. These calculations are addressed to basic dependent variables, rather than to the components of a tensor of effective material constants, which ensures considerable reduction of dimensionality as compared to the traditional G-closure approach.

References

1. LURIE, K. A., *The Extension of Optimization Problems Containing Controls in the Coefficients*, Proceedings of the Royal Society of Edinburgh, Vol. 114A, pp. 81-97, 1990.
2. LURIE, K. A., FEDOROV, A. V., and CHERKAEV, A. V., *Regularization of Optimal Design Problems for Bars and Plates, Parts 1 and 2*, Journal of Optimization Theory and Applications, Vol. 37, pp. 499-521, 1982 and Vol. 37, pp. 523-543, 1982.
3. LURIE, K. A., and CHERKAEV, A. V., *G-Closure of a Set of Anisotropically Conducting Media in the Two-Dimensional Case*, Journal of Optimization Theory and Applications, Vol. 42, pp. 283-304, 1984.
4. BALL, J. M., ... Archive for Rational Mechanics and Analysis, Vol. 63, pp. 337-403, 1977.
5. STRANG, G., *The Polyconvexification of $F(\nabla u)$*, Research Report CMA-R09-83, Australian National University, 1983.
6. KOHN, R. V., and STRANG, G., *Optimal Design and Relaxation of Variational Problems, Parts 1, 2, 3*, Communications on Pure and Applied Mathematics, Vol. 39, pp. 113-137, 1986; Vol. 39, pp. 139-182, 1986; and Vol. 39, pp. 353-377, 1986.

Direct Relaxation of Optimal Layout Problems for Plates[1]

K. A. Lurie[2]

Department of Mathematical Sciences

Worcester Polytechnic Institute

Worcester, MA 01609–2280

---

[1]This paper is dedicated to Professor Frithiof I. Niordson on the occasion of his 70th birthday. The research has been supported by AFOSR Grant No. 90–0268. The author acknowledges fruitful discussions with Robert P. Lipton.

[2]Professor, Department of Mathematical Sciences, Worcester Polytechnic Institute, Worcester, Massachusetts.

Abstract. The paper suggests an application of a direct procedure initiated in Ref. 1 to problems of optimal layout for plates. Optimal microstructures are explicitly indicated for a number of special cases, particularly, for the case when the original and conjugate strain tensors are coaxial.

Key Words. Direct relaxation, optimal microstructures, necessary conditions.

Introduction

In this paper we consider nonselfadjoint optimization problems for thin anisotropic plates subjected to transverse load. The state of equilibrium of such a plate is described by the equation

$$\nabla \cdot \nabla \cdot \mathscr{D} \cdot\cdot \nabla\nabla w = q, \quad (x,y)\in\Sigma \tag{1}$$

where w denotes the normal displacement, $\mathscr{D}$— the tensor of stiffness, and q the transverse load density. The boundary $\partial\Sigma$ of a plate will be assumed clamped, this property expressed by the boundary conditions

$$w\Big|_{\partial\Sigma} = \partial w/\partial n\Big|_{\partial\Sigma} = 0. \tag{2}$$

It will be assumed that $\mathscr{D} = \mathscr{D}(x,y)$ plays the role of control and may take one of two admissible values $\mathscr{D}_1$ or $\mathscr{D}_2$ at each point of the plate. The materials 1 and 2 with tensors $\mathscr{D}_1$ and $\mathscr{D}_2$ of stiffness will both be assumed isotropic, i.e.

$$\mathscr{D}_i = k_i a_1 a_1 + \mu_i(a_2 a_2 + a_3 a_3), \quad i = 1,2. \tag{3}$$

Here and below, a_1, a_2, a_3 represent an orthonormal basis in the space of 2nd rank symmetric tensors in the plane, i.e.

$$a_1 = (1/\sqrt{2})(ii+jj), \quad a_2 = (1/\sqrt{2})(ii-jj), \quad a_3 = (1/\sqrt{2})(ij+ij). \tag{4}$$

Introduce the characteristic function $\chi_1(x,y)$ of domain occupied by material 1 with tensor $\mathscr{D}_1$ of stiffness, and a similar function $\chi_2(x,y)$ for material 2; obviously, $\chi_1 + \chi_2 = 1$. It is required to find the distribution

$$\mathscr{D}(x,y) = \chi_1(x,y)\ \mathscr{D}_1 + \chi_2(x,y)\mathscr{D}_2 \qquad (5)$$

of the stiffness tensor throughout Σ which maximizes some weakly continuous functional $I(w)$ of solution to the boundary value problem (1), (2). Weak continuity is supposed to be with respect to $W_2^2(\Sigma)$, this space naturally associated with (1), (2). Specifically, as a typical example, we will consider the functional

$$I(w) = -\int_{\Sigma} [w(x,y) - w_0(x,y)]^2 dxdy$$

where $w_0(x,y) \in L_2(\Sigma)$.

This and similar optimization problems are known to be ill–posed and therefore requiring relaxation, i.e., the construction of an appropriate minimal extension of the initial set $U = \{\mathscr{D}_1, \mathscr{D}_2\}$ of admissible controls. Such an extension is currently offered on the basis of a precise knowledge of the G–closure of U, i.e. the set GU of invariants of the effective stiffness tensors $\mathscr{D}_0$ of all composites assembled from the elements of U (Ref. 2). However, the G–closures are known only for a few particular examples (Ref. 3), and the plate problem is not among them. Yet for these selected examples, the construction of GU represents difficulties, and for the plate problem these difficulties are still not overcome.

At the same time, for many applications we do not need to know the GU–set in full. Instead, it is often enough to specify some linear combination of components of $\mathscr{D}_0$; for our problem, this is the combination $\mathscr{D}_0 \cdot\cdot \nabla\nabla w$ which only matters in view of the Hooke's law. To determine this combination, we apply a *direct approach*, free from any reference to the G–closure.

Similar problems for the 2nd order equation $\nabla \cdot \mathscr{D} \cdot \nabla w = f$ have been discussed in Refs. 1,9.

2. Reduction to a sup inf problem

We first reduce the problem to a convenient sup inf form. Introduce the Lagrange multiplier λ and consider the augmented functional

$$J = J(w,\lambda) = I(w) + \int_{\Sigma} \lambda(\nabla\cdot\nabla\cdot\mathscr{D}\cdot\cdot\nabla\nabla w - q)dxdy, \tag{6}$$

the second member at the right–hand side taking into account the equation (1).

Equating to zero the first variation of (6) with respect to w and bearing (2) in mind, we arrive at the conjugate equation

$$\nabla\cdot\nabla\cdot\mathscr{D}\cdot\cdot\nabla\nabla\lambda = 2(w - w_0) \tag{7}$$

and boundary conditions

$$\lambda\Big|_{\partial\Sigma} = \partial\lambda/\partial n\Big|_{\partial\Sigma} = 0. \tag{8}$$

After integration by parts with the boundary conditions (8), the functional (6) takes on the form

$$J = I + \int_{\Sigma} (\nabla\nabla\lambda\cdot\cdot\mathscr{D}\cdot\cdot\nabla\nabla w - \lambda q)dxdy \tag{9}$$

convenient for a subsequent use.

The problem

$$\sup_{\mathscr{D},\,w} I$$

subjected to (1), (2) is equivalent to

$$\sup_{\mathscr{D},w}\ \inf_{\lambda} J \tag{10}$$

subjected to (2), (8). This is since by (6),

$$\inf_{\lambda} J = I + \inf_{\lambda}\int_{\Sigma} \lambda(\nabla\cdot\nabla\cdot \mathscr{D}\cdot\cdot\nabla\nabla w - q)dxdy =$$

$$I + \begin{cases} 0 & \text{if } \nabla\cdot\nabla\cdot\mathscr{D}\cdot\cdot\nabla\nabla w = q, \\ \infty & \text{otherwise.} \end{cases}$$

We observe that Eq. (1) appears as a necessary condition for a minimum in λ. Bearing (8) in mind, we may assume that J in (10) has the form (9). We have finally for (10)

$$\sup_{\mathscr{D},w}\ \inf_{\lambda}\{I+\int_{\Sigma} (\nabla\nabla\lambda\cdot\cdot\mathscr{D}\cdot\cdot\nabla\nabla w - \lambda q)dxdy\} \tag{11}$$

where $\mathscr{D}\in U = \{\mathscr{D}_1, \mathscr{D}_2\}$, and w and λ satisfy, respectively, equations (2) and (8).

In the sequel, we will establish the upper and lower bounds for the functional (11). An upper bound will be constructed analytically through an appropriate mathematical construction, and the lower bound will be generated by a specially chosen composite assembled from the original constituents. Both bounds will be shown to coincide, and desired relaxation will thus be achieved.

3. Upper bound for $\sup_{\mathscr{D},w}\ \inf_{\lambda} J$

This functional possesses the following upper bound:

$$\sup_{\mathscr{D},w}\inf_{\lambda} J = \sup_{w}\sup_{\mathscr{D}}\inf_{\lambda} J \le \sup_{w}\inf_{\lambda}\sup_{\mathscr{D}} J =$$

$$= \sup_{w}\inf_{\lambda}\left[-\int_{\Sigma}(w-w_0)^2 dxdy - \int_{\Sigma}\lambda q dxdy + \int_{\Sigma} G(\nabla\nabla w, \nabla\nabla\lambda)dxdy\right] \tag{12}$$

where

$$G(\xi,\eta) = \begin{cases} \xi \cdot\cdot \mathscr{D}_1 \cdot\cdot \eta \ , \ \xi \cdot\cdot \mathscr{D}_1 \cdot\cdot \eta \ge \xi \cdot\cdot \mathscr{D}_2 \cdot\cdot \eta, \\ \xi \cdot\cdot \mathscr{D}_2 \cdot\cdot \eta \ , \ \xi \cdot\cdot \mathscr{D}_1 \cdot\cdot \eta \le \xi \cdot\cdot \mathscr{D}_2 \cdot\cdot \eta. \end{cases} \tag{13}$$

The notation $\xi = \nabla\nabla w$, $\eta = \nabla\nabla\lambda$ will be used below. The function $G(\xi,\eta)$ is convex with respect to any of its arguments but non–convex with respect to their union.

The problem

$$\sup_{w}\inf_{\lambda}\left[-\int_{\Sigma}(w-w_0)^2 dxdy - \int_{\Sigma}\lambda q dxdy + \int_{\Sigma} G(\nabla\nabla w, \nabla\nabla\lambda)dxdy\right] \tag{14}$$

is still ill–posed. It would be well–posed if the integrand $G(\xi,\eta)$ were a saddle function, i.e. concave in ξ for fixed η and convex in η for fixed ξ. The solution would then exist and operations sup and inf commute. For our problem it is obviously not the case. However, the requirement that the function $G(\xi,\eta)$ be saddle is too restrictive now that ξ and η are gradients; to ensure the existence of sup inf for this case, it is enough to require that this function be only quasisaddle (Ref. 1). The quasisaddle envelope $G^{**}(\xi,\eta)$ of $G(\xi,\eta)$ will be constructed applying the so called polysaddlification transform introduced in Ref. 1. This transform plays the same role for sup inf problems as the polyconvexification transform (Refs. 4–6) plays for the minimum problems. For the fourth order problem considered, the

polysaddlification transform is given by the formula

$$G^{**}(\xi,\eta) = \sup_{\omega,d}\sup_{b}\inf_{a}\{a\cdot\cdot\xi+b\cdot\cdot\eta+\omega\cdot\cdot(\xi\times\eta)+d\xi\cdot\cdot T\cdot\cdot\eta$$

$$-\inf_{\xi}\sup_{\eta}[a\cdot\cdot\xi+b\cdot\cdot\eta+\omega\cdot\cdot(\xi\times\eta)+d\xi\cdot\cdot T\cdot\cdot\eta-G(\xi,\eta)]\} \qquad (15)$$

Here we introduced the notation T for a tensor

$$T = a_1a_1 - a_2a_2 - a_3a_3; \qquad (16)$$

the terms $\omega\cdot\cdot\xi\times\eta$ and $d\xi\cdot\cdot T\cdot\cdot\eta$ represent the null–Lagrangians $\xi\times\eta$ and $\xi\cdot\cdot T\cdot\cdot\eta$ (Refs. 3–6) taken into account with the aid of Lagrange multipliers ω and d.

The transform $G^{**}(\xi,\eta)$ defined by (15) satisfies the inequality

$$G^{**}(\xi,\eta) \geq G(\xi,\eta) \qquad (17)$$

for any $G(\xi,\eta)$ convex in η and arbitrary in ξ (Ref. 1).

Applying $G^{**}(\xi,\eta)$ instead of $G(\xi,\eta)$ we arrive at the upper bound

$$\sup_{w}\inf_{\lambda}[-\int_{\Sigma}(w-w_0)^2dxdy - \int_{\Sigma}\lambda qdxdy + \int_{\Sigma}G^{**}(\xi,\eta)dxdy] \qquad (18)$$

for (14), and, consequently, for the original functional (10).

4. Computation of $G^{**}(\xi,\eta)$

We first compute $\bar{H}(\xi,b) = \sup_{\eta}[b\cdot\cdot\eta - H(\xi,\eta)]$ with $H(\xi,\eta) = -\omega\cdot\cdot(\xi\times\eta) - d\xi\cdot\cdot T\cdot\cdot\eta + G(\xi,\eta)$:

$$b\cdot\cdot\eta - H(\xi,\eta) = \begin{cases} c^1\cdot\cdot\eta \text{ if } \eta \in \xi\cdot\cdot(\mathscr{D}_1-\mathscr{D}_2)\cdot\cdot\eta \geq 0, \\ c^2\cdot\cdot\eta \text{ if } \eta \in \xi\cdot\cdot(\mathscr{D}_1-\mathscr{D}_2)\cdot\cdot\eta \leq 0. \end{cases}$$

The tensors c^1, c^2 are defined as $(\mathrm{dev}\xi = \xi_2 a_2 + \xi_3 a_3)$

$$\begin{aligned} c^1 &= b + (d-k_1)\xi_1 a_1 - (d+\mu_1)\mathrm{dev}\xi + \omega \times \xi, \\ c^2 &= b + (d-k_2)\xi_1 a_1 - (d+\mu_2)\mathrm{dev}\xi + \omega \times \xi. \end{aligned} \tag{19}$$

By argument similar to that described in Ref. 1 we arrive at the formula

$$\bar{H}(\xi,b) = \sup_{\eta}[b\cdot\cdot\eta - H(\xi,\eta)] = \begin{cases} 0 & \text{if } b = \langle S\rangle\cdot\cdot\xi, \\ & \\ +\infty & \text{otherwise.} \end{cases} \tag{20}$$

In (20), the matrix $\langle S\rangle$ is defined as the convex hull

$$\langle S\rangle = t_1 S_1 + t_2 S_2, \quad t_1, t_2 \geq 0, \quad t_1 + t_2 = 1 \tag{21}$$

of matrices

$$S_i = \Delta_i + \omega\cdot\cdot\epsilon, \quad \Delta_i = \mathscr{D}_i - dT, \quad i = 1,2, \tag{22}$$

where the matrix

$$\epsilon = -E \times E \tag{23}$$

defines the Levi–Civita tensor of the 6th rank acting in the linear space of 2×2 symmetric tensors. The unit tensor E in this space can be defined as

$$E = a_1 a_1 + a_2 a_2 + a_3 a_3 \tag{24}$$

in the basis (4), and by a similar formula in any other orthonormal basis.

We make note of the formulas (Ref. 7)

$$\epsilon = -E \times E = -a_s a_s \times a_k a_k = -a_s a_t a_k \, \epsilon^{skt} = a_s a_t a_k \epsilon^{stk} \tag{25}$$

where

$$\epsilon^{skt} = a_s \cdot\cdot (a_k \times a_t) \tag{26}$$

are Levi–Civita symbols ($\epsilon^{123} = \epsilon^{231} = \epsilon^{312} = 1$, $\epsilon^{132} = \epsilon^{213} = \epsilon^{321} = -1$, $\epsilon^{stk} = 0$ otherwise); also

$$\omega \cdot\cdot \epsilon = -\omega \cdot\cdot E \times E = -\omega \times E = -E \times \omega = \epsilon \cdot\cdot \omega. \tag{27}$$

Geometrically, the function $\overline{h}(\xi, b)$ of ξ for fixed b is equal to positive infinity everywhere except for points of the set

$$b = \langle S \rangle \cdot\cdot \xi, \quad t_1, t_2 \in (21). \tag{28}$$

Equation (28) can be inverted to express ξ in terms of b. To this end we introduce symmetric tensors of the 4th rank (see (22))

$$\Delta_1 = \mathscr{D}_1 - dT, \quad \Delta_2 = \mathscr{D}_2 - dT, \quad \langle\Delta\rangle = t_1\Delta_1 + t_2\Delta_2 \tag{29}$$

and compute the inverse matrix $\langle S\rangle^{-1} = [\langle\Delta\rangle + \omega\cdot\cdot\epsilon]^{-1} = [\langle\Delta\rangle - \omega\times E]^{-1}$. We obtain by direct calculation

$$\langle S\rangle^{-1}=[1/(\det\langle\Delta\rangle+\omega\cdot\cdot\langle\Delta\rangle\cdot\cdot\omega)]\{(\det\langle\Delta\rangle)\langle\Delta\rangle^{-1}+\omega\omega+(\omega\cdot\cdot\langle\Delta\rangle)\times E\}=\delta+\Omega\times E \tag{30}$$

where

$$\delta = [1/(\det\langle\Delta\rangle+\omega\cdot\cdot\langle\Delta\rangle\cdot\cdot\omega)]\{(\det\langle\Delta\rangle)\langle\Delta\rangle^{-1}+\omega\omega\} \tag{31}$$

denotes the symmetric part of $\langle S\rangle^{-1}$ and

$$\Omega = [1/(\det\langle\Delta\rangle+\omega\cdot\cdot\langle\Delta\rangle\cdot\cdot\omega)](\omega\cdot\cdot\langle\Delta\rangle) \tag{32}$$

denotes the 2×2 tensor associated with its skew–symmetric part.

The set (28) is a segment of the curve in ξ–space traced as t_1 varies between 0 and 1. This segment connects points $\xi^{(1)}$ and $\xi^{(2)}$ corresponding, respectively, to $t_1 = 1$ and $t_1 = 0$:

$$\xi^{(1)} = S_1^{-1}\cdot\cdot b, \quad \xi^{(2)} = S_2^{-1}\cdot\cdot b. \tag{33}$$

We now compute the result of the operation

$$\inf_{a}\{a\cdot\cdot\xi-\inf_{\xi}[a\cdot\cdot\xi-(-\overline{h}(\xi,b))]\} \tag{34}$$

which comes second in the sequence (15). This one is known to put into correspondence with any given function $-\overline{h}(\xi,b)$ its concave ξ–envelope, i.e. the least concave function of ξ greater than or equal to $-\overline{h}(\xi,b)$. Particularly, if $-\overline{h}(\xi,b)$ is itself concave in ξ, then the operation (34) leaves this function intact.

In our special circumstances, this is obviously not the case. The concave envelope of $-\overline{h}(\xi,b)$ appears to be the function defined as negative infinity everywhere except for points of the convex hull Ξ of the curvilinear segment (28) where this envelope is equal to zero:

$$\inf_{a}\{a\cdot\cdot\xi-\inf_{\xi}[a\cdot\cdot\xi-(-\overline{h}(\xi,b))]\}=\begin{cases}0\ , & \xi\in\Xi\\ \\ -\infty\ , & \xi\notin\Xi.\end{cases} \tag{35}$$

The hull Ξ is a convex body in the ξ–space. We will assume that the curvilinear segment (28) and a line segment

$$(\xi_1-\xi_1^{(2)})/(\xi_1^{(1)}-\xi_1^{(2)})=(\xi_2-\xi_2^{(2)})/(\xi_2^{(1)}-\xi_2^{(2)})=(\xi_3-\xi_3^{(2)})/(\xi_3^{(1)}-\xi_3^{(2)}) \tag{36}$$

connecting the endpoints $\xi^{(1)}$ and $\xi^{(2)}$ (see (33)) both belong to the boundary $\partial\Xi$ of Ξ.

For our future purposes we need to know the left–hand side of (35) as the function of b for fixed ξ. This function can be defined as equal to negative infinity everywhere in the b–space except for the body $\mathcal{B}$ which appears as the "b–image" of Ξ, specifically, the boundary $\partial\mathcal{B}$ of $\mathcal{B}$ is described by the same equation as that of $\partial\Xi$, this time, however, ξ should be kept fixed whereas b should be considered variable. Obviously, the set (28) which is perceived as a *curvilinear* segment in the ξ–space appears as a *line* segment in the b–space, and in this capacity

belongs to $\partial\mathcal{R}$ Also, the set (36) which represents a *line* segment in the ξ–space appears as a *curvilinear* segment in the b–space, and this segment also belongs to $\partial\mathcal{R}$ Summarizing these results, we arrive at the following: the transform (15) reduces to a single operation

$$\sup_{\omega,d,b}[b\cdot\cdot\eta + \omega\cdot\cdot(\xi\times\eta) + d\xi\cdot\cdot T\cdot\cdot\eta] \tag{37}$$

subjected to the constraint $b \in \mathcal{R}$ Note that the set $\mathcal{R}$ itself depends on ω and d.

The curvilinear segment (36) in the b–space obviously represents a rib on $\partial\mathcal{R}$ The calculation (37) of the supremum with respect to b will include among others the possibility that the supremum is attained at points belonging to this segment. In the sequel, we investigate this possibility in major detail. Equation (36) can be represented in the equivalent form (see (33))

$$\xi = (m_1 S_1^{-1} + m_2 S_2^{-1}) \cdot\cdot b = \langle S^{-1}\rangle \cdot\cdot b. \tag{38}$$

Here, $m_1, m_2 \geq 0$, $m_1 + m_2 = 1$.

This relationship will be taken into account with the aid of the Lagrange multiplier Λ in the course of the maximization operation (37). We will examine stationary points of the function

$$\phi = b\cdot\cdot\eta + \omega\cdot\cdot(\xi\times\eta) + d\xi\cdot\cdot T\cdot\cdot\eta + \Lambda \cdot\cdot (\xi - \langle S^{-1}\rangle\cdot\cdot b) \tag{39}$$

viewed as the function of b,ω,d and m_1.

A routine calculation shows that

$$\phi_b = \eta - \Lambda \cdot\cdot \langle S^{-1}\rangle = 0$$

which means that

$$\Lambda = \eta \cdot\cdot \langle S^{-1} \rangle^{-1}. \tag{40}$$

With equations (38) and (40) in mind, the function ϕ becomes

$$\phi = \eta \cdot\cdot \langle S^{-1} \rangle^{-1} \cdot\cdot \xi + \omega \cdot\cdot (\xi \times \eta) + d\xi \cdot\cdot T \cdot\cdot \eta. \tag{41}$$

It can be shown (c.f. Ref. 7) that

$$\phi_\omega = -(\Lambda \cdot\cdot \langle S^{-1} \rangle \cdot\cdot b)_\omega + \xi \times \eta = m_1(\Lambda \cdot\cdot S_1^{-1}) \times (S_1^{-1} \cdot\cdot b)$$

$$+ m_2(\Lambda \cdot\cdot S_2^{-1}) \times (S_2^{-1} \cdot\cdot b) + \xi \times \eta.$$

This expression can be rewritten in either of two forms:

$$\phi_\omega = m_1(\Lambda \cdot\cdot S_1^{-1}) \times (S_1^{-1} \cdot\cdot b) + m_2(\Lambda \cdot\cdot S_2^{-1}) \times (S_2^{-1} \cdot\cdot b)$$

$$+ ((m_1 S_1^{-1} + m_2 S_2^{-1}) \cdot\cdot b) \times (\Lambda \cdot\cdot (m_1 S_1^{-1} + m_2 S_2^{-1})) = \tag{42}$$

$$= -m_1 m_2 (\Delta S^{-1} \cdot\cdot b) \times (\Lambda \cdot\cdot \Delta S^{-1}); \quad \Delta S^{-1} = S_2^{-1} - S_1^{-1},$$

or

$$\phi_\omega = \eta \cdot\cdot \langle S^{-1} \rangle^{-1} \cdot\cdot [m_1 S_1^{-1} \times S_1^{-1} + m_2 S_2^{-1} \times S_2^{-1}] \cdot\cdot \langle S^{-1} \rangle^{-1} \cdot\cdot \xi + \xi \times \eta. \tag{43}$$

The stationarity condition $\phi_\omega = 0$ can now be written as

$$\Delta S^{-1} \cdot\cdot b = (\Delta S^{-1}) \cdot\cdot \langle S^{-1} \rangle^{-1} \cdot\cdot \xi = \kappa \Lambda \cdot\cdot \Delta S^{-1} = \tag{44}$$

$$= \kappa \eta \cdot\cdot \langle S^{-1} \rangle^{-1} \cdot\cdot (\Delta S^{-1})$$

where κ is a scalar multiplier. An equivalent representation is associated with equation (43):

$$\eta \cdot\cdot \langle S^{-1}\rangle^{-1} \cdot\cdot [m_1 S_1^{-1} \times S_1^{-1} + m_2 S_2^{-1} \times S_2^{-1}] \cdot\cdot \langle S^{-1}\rangle^{-1} \cdot\cdot \xi + \xi \times \eta = 0 \qquad (45)$$

Condition $\phi_d = 0$ reduces to

$$\phi_d = -m_1 m_2 (\Delta S^{-1} \cdot\cdot b) \cdot\cdot T \cdot\cdot (\Lambda \cdot\cdot \Delta S^{-1}) = -m_1 m_2 \kappa^{-1} (\Delta S^{-1} \cdot\cdot b) \cdot\cdot T \cdot\cdot (\Delta S^{-1} \cdot\cdot b) = 0. \qquad (46)$$

or, equivalently,

$$\phi_d = -\eta \cdot\cdot \langle S^{-1}\rangle^{-1} \cdot\cdot [m_1 S_1^{-1} \cdot\cdot T \cdot\cdot S_1^{-1} + m_2 S_2^{-1} \cdot\cdot T \cdot\cdot S_2^{-1}] \cdot\cdot \langle S^{-1}\rangle^{-1} \cdot\cdot \xi + \xi \cdot\cdot T \cdot\cdot \eta = 0. \qquad (47)$$

Note that the stationarity condition (46) applies as the necessary condition for a maximum if the corresponding root d is such that the function ϕ defined by (41) is concave in d for all ω. To guarantee this, we must require that $\det S_i \geq 0 (i = 1,2)$, i.e. that

$$\det \Delta_i + \omega \cdot\cdot \Delta_i \cdot\cdot \omega \geq 0, \quad i = 1,2. \qquad (48)$$

These inequalities should be considered as additional constraints influencing the d–maximization.

Computing the expression (41) for ϕ at the stationary values of ω and d we have to maximize it with regard to m_1. Before we do so we investigate this expression in terms of its attainability with the aid of special microstructures. This is a right time for such investigation since the aforementioned construction explicitly depends on m_1, this dependence being very special for a number of popular microstructures.

After maximization in m_1, the expression (41) should produce a final construction (37) for $G^{**}(\xi,\eta)$. This program is elaborate in its entirety, and we will begin with the analysis of several special cases.

5. Case when tensors ξ and η are proportional

In this case, the assumption $\omega = \Omega = 0$ obviously satisfies equation (45) since the matrices S_1, S_2, $\langle S\rangle$ and $\langle S^{-1}\rangle$ are then symmetric. Equation (46) is reduced to

$$(\Delta\delta\cdot\cdot\langle\delta\rangle^{-1}\cdot\cdot\xi)\cdot\cdot T\cdot\cdot(\Delta\delta\cdot\cdot\langle\delta\rangle^{-1}\cdot\cdot\xi) = 0 \tag{49}$$

where (c.f. Eq. (31))

$$\Delta\delta = \delta_2-\delta_1 = \Delta_2^{-1} - \Delta_1^{-1},\ \langle\delta\rangle^{-1} = (m_1\Delta_1^{-1}+m_2\Delta_2^{-1})^{-1}.$$

The tensors $\Delta_i (i = 1,2)$ are defined by Eqs. (29), (3) and (16) as

$$\Delta_i = K_i a_1 a_1 + M_i(a_2a_2+a_3a_3),\ K_i = k_i-d,\ M_i = \mu_i + d. \tag{50}$$

We therefore obtain

$$\Delta\delta\cdot\cdot\langle\delta\rangle^{-1}\cdot\cdot\xi = [\Delta K^{-1}a_1a_1+\Delta M^{-1}(a_2a_2+a_3a_3)]\cdot\cdot[\langle K^{-1}\rangle^{-1}a_1a_1+\langle M^{-1}\rangle^{-1}(a_2a_2+a_3a_3)]\cdot\cdot\xi$$

$$= (\Delta K^{-1})\langle K^{-1}\rangle^{-1}\xi_1a_1 + (\Delta M^{-1})\langle M^{-1}\rangle^{-1}(\xi_2a_2+\xi_3a_3) \tag{51}$$

where

$$\Delta K^{-1} = K_2^{-1} - K_1^{-1},\ \Delta M^{-1} = M_2^{-1} - M_1^{-1},$$

$$\langle K^{-1}\rangle^{-1} = (m_1K_1^{-1}+m_2K_2^{-1})^{-1},\ \langle M^{-1}\rangle^{-1} = (m_1M_1^{-1}+m_2M_2^{-1})^{-1}. \tag{52}$$

Equation (49) shows that the second invariant of (51) equals zero, i.e.

$$(\Delta K^{-1})^2\langle K^{-1}\rangle^{-2}\xi_1^2 = (\Delta M^{-1})^2\langle M^{-1}\rangle^{-2}(\xi_2^2+\xi_3^2).$$

Introducing the ratio

$$\zeta = |\mathrm{dev}\xi|/\xi_1 = \sqrt{(\xi_2^2+\xi_3^2)}/\xi_1 \tag{53}$$

of deviatoric and spherical parts of tensor ξ, we arrive at the equation

$$\zeta^2 = \left[\frac{(k_2-k_1)(d+m_1\mu_2+m_2\mu_1)}{(\mu_2-\mu_1)(d-m_1k_2-m_2k_1)}\right]^2$$

defining the Lagrange multiplier d

$$d = (\zeta\tilde{k}\Delta\mu-\tilde{\mu}\Delta k)/(\zeta\Delta\mu+\Delta k), \tag{54}$$

$$\tilde{k} = m_1k_2+m_2k_1, \quad \tilde{\mu} = m_1\mu_2 + m_2\mu_1,$$

$$\Delta k = k_2-k_1, \ \Delta\mu = \mu_2 - \mu_1. \tag{55}$$

Equation (54) has been obtained earlier by Gibianskii and Cherkaev in Ref. 8. We use (54) to eliminate d from the expression (39), the resulting construction is attainable by a laminar composite of the lst rank (Ref. 8).

6. Case when tensors ξ and η are coaxial

This case generalizes the previous one but is related to a new situation when we *cannot* apply the G–closure technique (Ref. 3) to construct the required relaxation; on the contrary, the case of Section 5 is self–adjoint and therefore can be handled with the aid of such technique. In the new circumstances, no G–closure is known, and the direct method demonstrates here its genuine power.

Because the tensors ξ and η are coaxial, we can choose the basis a_1, a_2, a_3 (see (4)) so that

$$\xi = \xi_1 a_1 + \xi_2 a_2, \tag{56}$$

$$\eta = \eta_1 a_1 + \eta_2 a_2; \tag{57}$$

the tensor ω will be assumed having only a_3–component, namely

$$\omega = \omega_3 a_3. \tag{58}$$

Direct calculation of the matrix $\langle S^{-1}\rangle^{-1}$ shows that

$$\langle S^{-1}\rangle^{-1} = Z^{-1}(PQ+\omega_3^2 R^2)^{-1}(QZa_1a_1 + PZa_2a_2 + PQa_3a_3 + \omega_3^2 R^2 a_3a_3 - \omega_3 RZa_3{\times}E) \tag{59}$$

where

$$P = \left\langle M/(KM+\omega_3^2)\right\rangle,\ Q = \left\langle K/(KM+\omega_3^2)\right\rangle,\ R = \left\langle 1/(KM+\omega_3^2)\right\rangle,\ Z = \langle 1/M\rangle, \tag{60}$$

and symbol $\langle\cdot\rangle$ denotes averaging, i.e., for example,

$$\langle 1/M\rangle = m_1/M_1 + m_2/M_2 = m_1/(\mu_1+d) + m_2/(\mu_2+d), \tag{61}$$

etc.

The tensor b computed as $b = \langle S^{-1}\rangle^{-1} \cdot\cdot\, \xi$ (c.f. (38)) turns out to be coaxial with ξ because of (56) and (59):

$$b = \langle S^{-1}\rangle^{-1} \cdot\cdot\, \xi = [1/(PQ+\omega_3^2R^2)]\Big[(Q\xi_1+\omega_3 R\xi_2)a_1 + (P\xi_2-\omega_3 R\xi_1)a_2\Big]. \tag{62}$$

The matrix ΔS^{-1} can easily be computed, too; this one equals

$$\Delta S^{-1} = pa_1a_1 + qa_2a_2 + za_3a_3 + \omega_3 ra_3{\times}E \tag{63}$$

where

$$p = \Delta(M/(KM+\omega_3^2)),\ q = \Delta(K/(KM+\omega_3^2)),\ r = \Delta(1/(KM+\omega_3^2)),\ z = \Delta(1/M), \tag{64}$$

and symbol $\Delta(\cdot)$ denotes the difference, i.e. for example,

$$\Delta(1/M) = 1/M_2 - 1/M_1, \tag{65}$$

etc.

The tensor $\Delta S^{-1} \cdot\cdot\, b = \Delta S^{-1} \cdot\cdot\, \langle S^{-1}\rangle^{-1} \cdot\cdot\, \xi$ is now computed as

$$\Delta S^{-1} \cdot\cdot\, b = [1/(PQ + \omega_3^2R^2)]\{[(pQ + \omega_3^2 rR)\xi_1 + \omega_3(pR - rP)\xi_2]a_1 +$$

$$[(qP + \omega_3^2 rR)\xi_2 - \omega_3(qR - rQ)\xi_1]a_2\}.$$

A similar formula for $\Lambda\cdot\cdot\Delta S^{-1} = \eta\cdot\cdot\langle S^{-1}\rangle^{-1}\cdot\cdot\Delta S^{-1}$ is given by

$$\Lambda\cdot\cdot\Delta S^{-1} = [1/(PQ+\omega_3^2R^2)]\{[(pQ+\omega_3^2rR)\eta_1 - \omega_3(pR-rP)\eta_2]a_1$$

$$+ [(qP+\omega_3^2rR)\eta_2 + \omega_3(qR-rQ)\eta_1]a_2\}.$$

Direct calculation shows that $(B_i = K_iM_i^2+\omega_3^2,\ i = 1,2)$

$$-(\Delta S^{-1}\cdot\cdot b)(PQ+\omega_3^2R^2)B_1B_2 = (\tilde{M}\Delta k\xi_1-\omega_3\Delta\mu\xi_2)a_1 + (\tilde{K}\Delta\mu\xi_2 + \omega_3\Delta k\xi_1)a_2, \quad (66)$$

$$-(\Lambda\cdot\cdot\Delta S^{-1})(PQ+\omega_3^2R^2)B_1B_2 = (\tilde{M}\Delta k\eta_1+\omega_3\Delta\mu\eta_2)a_1 + (\tilde{K}\Delta\mu\eta_2 - \omega_3\Delta k\eta_1)a_2, \quad (67)$$

$$\tilde{K} = m_1K_2 + m_2K_1, \quad \tilde{M} = m_1M_2 + m_2M_1. \quad (68)$$

We are now ready to apply the necessary conditions (44) and (46). The first of them is reduced to

$$(\omega_3^2-\overline{MK})(\xi_1\eta_2-\xi_2\eta_1) + 2\omega_3(\overline{K}\xi_2\eta_2 + \overline{M}\xi_1\eta_1) = 0, \quad (69)$$

$$K = \tilde{K}\ \Delta\mu/\Delta k, \quad \overline{M} = \tilde{M}\ \Delta k/\Delta\mu\ , \ \overline{KM} = \tilde{K}\tilde{M}.$$

In view of (44), Eq. (46) can be rewritten as

$$(\Delta S^{-1}\cdot\cdot b)\cdot\cdot T\cdot\cdot(\Delta S^{-1}\cdot\cdot b) = 0. \quad (70)$$

Combining this with (66) we get

$$\xi_2/\xi_1 = [(\tilde{M} \mp \omega_3)/(\omega_3 \pm \tilde{K})](\Delta k/\Delta\mu). \quad (71)$$

Equations (69) and (71) comprise a system that can be solved to determine ω_3 and d; we obtain ($\zeta = \xi_2/\xi_1$, $\sigma = \eta_2/\eta_1$)

$$\omega_3 = ((\bar{k}+\bar{\mu})/2)\Delta k\Delta\mu(\sigma-\zeta)/[(\sigma\Delta\mu\pm\Delta k)(\zeta\Delta\mu\pm\Delta k)] \tag{72}$$

and

$$d=[1/2(\sigma\Delta\mu\pm\Delta k)(\zeta\Delta\mu\pm\Delta k)][2\sigma\zeta\bar{k}(\Delta\mu)^2\pm\Delta\mu\Delta k(\sigma+\zeta)(\bar{k}-\bar{\mu})-2\bar{\mu}(\Delta k)^2]. \tag{73}$$

Equations (72), (73) provide a basis for the subsequent final calculations. We compute the bilinear form (41) making use of (72), (73). Direct calculation shows that

$$\phi = \phi_{1\pm} = \eta\cdot\cdot\langle\mathscr{D}\rangle\cdot\cdot\xi - [m_1m_2/(\bar{k}+\bar{\mu})](\xi_1\Delta k\pm\xi_2\Delta\mu)(\eta_1\Delta k\pm\eta_2\Delta\mu) \tag{74}$$

where

$$\langle\mathscr{D}\rangle = m_1\mathscr{D}_1 + m_2\mathscr{D}_2.$$

The values (74) of $\phi_{1\pm}$ is attained by the rank 1 laminate with layers parallel to the main axes of tensor a_2, i.e. the main axes of ξ and η.

This regime will be valid within the range of parameters $\zeta = \xi_2/\xi_1$, $\sigma = \eta_2/\eta_1$ defined by Ineqs. (48) together with (72), (73). Without the range, rank 2 laminates will be applied to saturate the corresponding bounds.

To show this, consider for example, the case $\det\Delta_2 + \omega\cdot\cdot\Delta_2\cdot\cdot\omega = 0$ or, in view of (58),

$$B_2 = K_2M_2 + \omega_3^2 = 0. \tag{75}$$

This is a manifold in the space (ω_3,d), and the variations $\delta\omega = a_3\delta\omega_3$, δd are therefore linked by the relationship (see (50))

$$2d\delta d - (k_2 - \mu_2)\delta d - 2\omega_3\delta\omega_3 = 0$$

as we move along this manifold. The latter relation can be rewritten as (see (50))

$$\delta d = 2\omega_3\delta\omega_3/(M_2-K_2), \tag{76}$$

and instead of two necessary conditions $\phi_\omega = \phi_d = 0$ (see (44) and (46)), we arrive at only one condition

$$(\Delta S^{-1}\cdot\cdot b)\times(\Lambda\cdot\cdot\Delta S^{-1})\cdot\cdot a_3\delta\omega_3 + (\Delta S^{-1}\cdot\cdot b)\cdot\cdot T\cdot\cdot(\Lambda\cdot\cdot\Delta S^{-1})\ 2\omega_3\delta\omega_3/(M_2-K_2) = 0$$

or, equivalently,

$$(\Delta S^{-1}\cdot\cdot b)\times(\Lambda\cdot\cdot\Delta S^{-1})\cdot\cdot a_3 + [2\omega_3/(M_2-K_2)](\Delta S^{-1}\cdot\cdot b)\cdot\cdot T\cdot\cdot(\Lambda\cdot\cdot\Delta S^{-1}) = 0. \tag{77}$$

This condition should hold along with (75).

Equation (77) can be transformed with the aid of equations (66), (67) defining matrices $\Delta S^{-1}\cdot\cdot b$ and $\Lambda\cdot\cdot\Delta S^{-1}$. We arrive at the relationship

$$\left[\tilde{M}\tilde{K} - \omega_3^2 + 2\omega_3^2(\tilde{M}-\tilde{K})/(M_2-K_2)\right](\sigma-\zeta) + 2\omega_3\{[(\tilde{M}^2-K_2M_2)/(M_2-K_2)-\tilde{M}](\Delta k/\Delta\mu)$$

$$-[(\tilde{K}^2-K_2M_2)/(M_2-K_2) + \tilde{K}](\Delta\mu/\Delta k)\sigma\zeta\} = 0. \tag{78}$$

Expressions in the square brackets can be transformed as we use (75) to eliminate ω_3^2. After some algebra we arrive at the relationships

$$\bar{M}\bar{K}-\omega_3^2+2\omega_3^2(\bar{M}-\bar{K})/(M_2-K_2) = [m_2/(M_2-K_2)](\beta d+\gamma),$$

$$[(\bar{M}^2-K_2M_2)/(M_2-K_2)-\bar{M}](\Delta k/\Delta\mu)-[(\bar{K}^2-K_2M_2)/(M_2-K_2)+\bar{K}](\Delta\mu/\Delta k)\sigma\zeta=-m_2c/(M_2-K_2).) \tag{79}$$

Here, symbols β, γ and c are defined as

$$\begin{aligned} \beta &= -(u+v), \\ \gamma &= k_2v-\mu_2u, \\ c &= u-v\sigma\zeta, \end{aligned} \tag{80}$$

where

$$u = (k_2+\bar{\mu})\Delta k,\ \ v = (\bar{k}+\mu_2)\Delta\mu. \tag{81}$$

Eq. (78) now shows that

$$\omega_3 = (1/2c)(\beta d+\gamma)(\sigma-\zeta). \tag{82}$$

We now use this relation to eliminate ω_3 from (75). The result will be quadratic equation for d:

$$d^2[\beta^2(\sigma-\zeta)^2 - 4c^2] + 2[\beta\gamma(\sigma-\zeta)^2 + 2c^2(k_2 - \mu_2)]d$$

$$+ \gamma^2(\sigma-\zeta)^2 + 4c^2k_2\mu_2 = 0. \tag{83}$$

The discriminant of this equation is equal to

$$4c^2\{(\sigma-\zeta)^2(\gamma+\beta k_2)(\gamma-\beta\mu_2) + c^2(k_2+\mu_2)^2\}.$$

From Eqs. (80), (82) it follows that

$$(\gamma+\beta k_2)(\gamma-\beta\mu_2) = -uv(k_2+\mu_2)^2$$

and the discriminant turns out to be

$$4c^2(k_2+\mu_2)^2[-(\sigma^2 - 2\sigma\zeta + \zeta^2)uv + u^2 - 2uv\sigma\zeta + v^2\sigma^2\zeta^2]$$
$$= 4v^2c^2(k_2+\mu_2)^2(\sigma^2 - u/v)(\zeta^2 - u/v)\,.$$

Eq. (83) now shows that

$$d = -\{1/[\beta^2(\sigma-\zeta)^2 - 4c^2]\}[\beta\gamma(\sigma-\zeta)^2 + 2c^2(k_2-\mu_2)$$

$$\mp 2vc(k_2+\mu_2)\sqrt{(\sigma^2 - u/v)(\zeta^2 - u/v)}]. \tag{84}$$

The corresponding values of ω_3 will be

$$\omega_3 = \{(\sigma-\zeta)/[\beta^2(\sigma-\zeta)^2 - 4c^2]\}\{-\beta c(k_2-\mu_2) - 2\gamma c \pm \beta v(k_2+\mu_2)\sqrt{(\sigma^2-u/v)(\zeta^2-u/v)}\}$$

or, in view of (80),

$$\omega_3 = -\{(k_2 + \mu_2)(\sigma - \zeta)/[\beta^2(\sigma - \zeta)^2 - 4c^2]\}\{c(v - u) \pm v(u + v)\sqrt{(\sigma^2 - u/v)(\zeta^2 - u/v)}\}. \qquad (85)$$

Now it is easy to compute the bilinear form (41). After some algebra we obtain

$$\phi/\xi_1\eta_1 = [(K_2M_2 - K_1M_1)/(K_2\Delta\mu + \bar{M}\Delta k)][K_2 + M_2\sigma\zeta - \omega_3(\sigma - \zeta)] +$$

$$+ \omega_3(\sigma - \zeta) + d(1 - \sigma\zeta).$$

Making use of (50) and (68), we reduce this to the form

$$\phi/\xi_1\eta_1 = k_2 + \mu_2\sigma\zeta + \{m_1\Delta k\Delta\mu/[k_2\Delta\mu + \bar{\mu}\Delta k + d(\Delta k - \Delta\mu)]\}$$

$$[-k_2 + d + \omega_3(\sigma - \zeta) - (d + \mu_2)\sigma\zeta]. \qquad (86)$$

With the aid of (84) and (85) one can show that

$$-k_2 + d + \omega_3(\sigma - \zeta) - (d + \mu_2)\sigma\zeta = \{(k_2 + \mu_2)/[(u + v)^2(\sigma - \zeta)^2 - 4c^2]\}\mathscr{L}$$

$$k_2\Delta\mu + \tilde{\mu}\Delta k + d(\Delta k - \Delta\mu) = \{1/[(u+v)^2(\sigma-\zeta)^2 - 4c^2]\}\mathscr{M},$$

where

$$\mathscr{L} = 2(1+\sigma\zeta)[c^2-uv(\sigma-\zeta)^2] \pm v[2c(1-\sigma\zeta)-(u+v)(\sigma-\zeta)^2]\sqrt{(\sigma^2- u/v)(\zeta^2- u/v)},$$

$$\mathscr{M} = 2uv(u+v)(\sigma-\zeta)^2-2c\left[2cu-(u-v)\left[c\pm v\sqrt{(\sigma^2- u/v)(\zeta^2- u/v)}\right]\right].$$

Now it is easy to check by direct inspection that

$$\mathscr{L}/\mathscr{M} = -[(k_2+\mu_2)/u]\left[u/v + \sigma\zeta \pm \sqrt{(\sigma^2- u/v)(\zeta^2- u/v)}\right],$$

and from (86) we obtain

$$\phi/\xi_1\eta_1 = \phi_{2\pm}/\xi_1\eta_1 = k_2 + \mu_2\sigma\zeta - [m_1\Delta k\Delta\mu(k_2 + \mu_2)/2]$$

$$\left[1/v + (\sigma\zeta/u) \pm (1/u)\sqrt{(\sigma^2-u/v)(\zeta^2-u/v)}\right]. \qquad (87)$$

The values (87) of $\phi_{2\pm}$ are attained by the rank 2 lamination with material $\mathscr{D}_1$ being the core and layers being parallel to the main axes of ξ and η. To show this, consider the formula

$$\mathscr{D}_0 = \mathscr{D}_2 + m_1[(\mathscr{D}_1-\mathscr{D}_2)^{-1} + [2m_2/(k_2+\mu_2)](\alpha_1 nnnn + \alpha_2 tttt)]^{-1} = \mathscr{D}_2 + m_1 A^{-1} \qquad (88)$$

for the effective tensor $\mathscr{D}_0$ of such a composite assembled from materials $\mathscr{D}_1$ and $\mathscr{D}_2$ taken with volume fractions m_1 and m_2, respectively. Parameters α_1, $\alpha_2 \geq 0$ ($\alpha_1 + \alpha_2 = 1$) are linked with the geometric parameters f,ρ of microstructure (see Figure) by the formulas

$$\alpha_1 = f(1-\rho)/m_2, \quad \alpha_2 = \rho/m_2.$$

The matrix A in (88) can be represented in the form

$$A = \pi\, a_1 a_1 + \theta(a_1 a_2 + a_2 a_1) + \theta a_2 a_2 + \tau a_3 a_3$$

where

$$\pi = -(\bar{k}+\mu_2)/[(k_2+\mu_2)\Delta k] = -v/[(k_2+\mu_2)\Delta k \Delta\mu],$$

$$\theta = m_2(2\alpha_1 - 1)/(k_2+\mu_2), \tag{89}$$

$$\rho = -(k_2+\bar{\mu})/[(k_2+\mu_2)\Delta\mu] = -u/[(k_2+\mu_2)\Delta k \Delta\mu],$$

$$\tau = -1/\Delta\mu,$$

and the basis a_1, a_2, a_3 is chosen as suggested in (4) and (56), (57) with the unit vectors i, j oriented along the main axes of ξ and η.

The inverse matrix A^{-1} is computed as

$$A^{-1} = (\rho/\chi)a_1 a_1 - (\theta/\chi)(a_1 a_2 + a_2 a_1) + (\pi/\chi)a_2 a_2 + (1/\tau)a_3 a_3$$

where χ is defined by the formula

$$\chi = \pi\rho - \theta^2.$$

The bilinear form $\xi \,..\, \mathscr{D}_0 \,..\, \eta$ obviously depends on α_1; the extremal values of this parameter can be found from the relationship

$$(\xi..A^{-1}..\eta)_{\alpha_1} = \xi..(A^{-1})_{\alpha_1}..\eta = 0$$

or, equivalently, from

$$\xi..A^{-1}..A_{\alpha_1}..A^{-1}..\eta = 0.$$

This one is easily reduced to

$$(\theta^2 + \pi\rho)(\xi_1\eta_2 + \xi_2\eta_1) - 2\theta(\rho\xi_1\eta_1 + \pi\xi_2\eta_2) = 0,$$

and we obtain the extremal values of θ

$$\theta = [\pi/(\sigma + \zeta)]\left[(\rho/\pi) + \sigma\zeta \pm \sqrt{(\sigma^2 - \rho/\pi)(\zeta^2 - \rho/\pi)}\right] \tag{90}$$

(recall that $\xi = \xi_2/\xi_1$ and $\sigma = \eta_2/\eta_1$).

With these values for θ it is easy to arrive at the following expression for the bilinear form

$$\xi\cdot\cdot\mathscr{D}_0\cdot\cdot\eta/\xi_1\eta_1 = k_2 + \mu_2\sigma\zeta + (m_1/2)\left[(1/\pi) + \sigma\zeta/\rho \pm (1/\rho)\sqrt{(\sigma^2 - \rho/\pi)(\zeta^2 - \zeta/\pi)}\right]$$

or, in view of (89)

$$\xi\cdot\cdot\mathscr{D}_0\cdot\cdot\eta/\xi_1\eta_1 = k_2 + \mu_2\sigma\zeta - [m_1\Delta k\Delta\mu(k_2+\mu_2)/2]\left[(1/v) + \sigma\zeta/u \pm (1/u)\sqrt{\sigma^2-u/v)(\zeta^2-u/v)}\right].$$

This expression is the same as (87), and the attainability of the latter bound is thereby proved. A

result similar to (87) can be established if the condition

$$B_1 = K_1 M_1 + \omega_3^2 = 0 \tag{91}$$

holds instead of (75). We then arrive at the formula

$$\phi/\xi_1\eta_1 = \phi_{3\pm}/\xi_1\eta_1 = k_1 + \mu_1\sigma\zeta + [m_2\Delta k\Delta\mu(k_1+\mu_1)/2]$$

$$\left[(1/\bar{v}) + \sigma\zeta/\bar{u} \pm (1/\bar{u})\sqrt{(\sigma^2 - \bar{u}/\bar{v})(\zeta^2 - \bar{u}/\bar{v})}\right] \tag{92}$$

with $\bar{u}$, $\bar{v}$ defined as (cf. (81)).

$$\bar{u} = (k_1 + \tilde{\mu})\Delta k, \quad \bar{v} = (\tilde{k} + \mu_1)\Delta\mu. \tag{93}$$

The values (92) are attained for the 2nd rank lamination with material $\mathcal{A}_2$ being the core and layers parallel to the main axes of ξ and η. Now it is easy to specify the ranges of parameters $\xi_1,\eta_1,\sigma,\zeta$ that maximize the function ϕ with respect to ω_3 and d (see (37) and (39)). We will consider, without loss of generality, the following two possibilities:

(i) $\Delta k > 0$, $\Delta\mu > 0$ – the "well ordered " case,

(ii) $\Delta k < 0$, $\Delta\mu > 0$ – the "badly ordered" case.

Case (i) – "well ordered".

Let us first select between regimes $(74)_+$ and $(74)_-^1$. We obtain

$$\phi_{1+} - \phi_{1-} = -[m_1m_2/(\tilde{k}+\tilde{\mu})](\Delta k\Delta\mu)(\xi_1\eta_2+\xi_2\eta_1) = -[m_1m_2/(\tilde{k}+\tilde{\mu})](\Delta k\Delta\mu)\xi_1\eta_1(\sigma+\xi)$$

which means that if $\xi_1\eta_1 \leq 0$, then

$$\max(\phi_{1+},\phi_{1-}) = \begin{cases} \phi_{1+} & \text{if} \quad \sigma + \zeta \geq 0, \\ \phi_{1-} & \text{if} \quad \sigma + \zeta \leq 0. \end{cases} \tag{94}$$

Also, if $\xi_1\eta_1 \geq 0$, then

$$\max(\phi_{1+},\phi_{1-}) = \begin{cases} \phi_{1-} & \text{if} \quad \sigma + \zeta \geq 0, \\ \phi_{1+} & \text{if} \quad \sigma + \zeta \leq 0. \end{cases} \tag{95}$$

The regimes (74) may neighbor either (87) or (92). Consider the case when the ranges $(74)_+$ and (92) have the common interface; both σ and ζ will be assumed continuous across this surface. The values of $\phi/\xi_1\eta_1$ should be equal at the interface, i.e.

$$\langle k\rangle + \langle\mu\rangle\sigma\zeta - [m_1m_2/(\bar{k}+\bar{\mu})](\Delta k + \xi\Delta\mu)(\Delta k + \sigma\Delta\mu) =$$

$$= k_1+\mu_1\sigma\zeta + [m_2\Delta k\Delta\mu(k_1+\mu_1)/2]\left[(1/\overline{v}) + \sigma\zeta/\overline{u} \pm (1/\overline{u})\sqrt{(\sigma^2 - \overline{u}/\overline{v})(\zeta^2 - \overline{u}/v)}\right]$$

or, equivalently,

---

[1]We apply the subscript "+" to designate the regime (74) related to the upper sign. This and similar agreements apply to other regimes as well.

$$(\overline{u} + \overline{v}\sigma\zeta)/(\bar{k} + \bar{\mu}) - m_1\Delta k\Delta\mu(\sigma + \zeta)/(\bar{k} + \bar{\mu}) =$$

$$[\Delta k\Delta\mu(k_1 + \mu_1)/2\overline{u\,v}]\left[\overline{u} + \overline{v}\sigma\zeta \pm \overline{v}\sqrt{(\sigma^2 - \overline{u}/\overline{v})(\zeta^2 - \overline{u}/\overline{v})}\right].$$

Another form of the latter equation is given by

$$(\overline{u} + \overline{v}\sigma\zeta)\left[(\overline{u\,v}/\Delta k\Delta\mu) + m_1^2\Delta k\Delta\mu\right] - 2m_1\overline{uv}(\sigma + \zeta) =$$

$$= \pm\,\overline{v}\left[(\overline{u\,v}/\Delta k\Delta\mu) - m_1^2\Delta k\Delta\mu\right]\sqrt{(\sigma^2 - \overline{u}/\overline{v})(\zeta^2 - \overline{u}/\overline{v})}. \qquad (96)$$

Here, we made use of the relationship

$$(k_1 + \bar{\mu})(\bar{k} + \mu_1) = (k_1 + \mu_1)(\bar{k} + \bar{\mu}) + m_1^2\Delta k\Delta\mu.$$

If we square both sides of Eq. (96), then after some algebra we arrive at

$$[(\overline{u\,v}/\Delta k\Delta\mu) + m_1^2\Delta k\Delta\mu](\sigma + \zeta) - 2m_1(\overline{u} + \overline{v}\sigma\zeta) = 0$$

or, equivalently,

$$[\zeta - \overline{u}/(m_1\Delta k\Delta\mu)](\sigma - m_1\Delta k\Delta\mu/\overline{v})$$

$$+(\zeta - m_1\Delta k\Delta\mu/\overline{v})[\sigma - \overline{u}(/m_1\Delta k\Delta\mu)] = 0\,. \qquad (97)$$

This equation represents a hyperbola in the $\sigma\zeta$–plane; it will be convenient to introduce coordinates

$$x_1 = \sigma + \zeta, \quad x_2 = \sigma - \zeta$$

and parameters

$$\overline{\alpha}_1 = \overline{u}/(m_1 \Delta k \Delta\mu) + m_1 \Delta k \Delta\mu/\overline{v}, \quad \overline{\alpha}_2 = \overline{u}/(m_1 \Delta k \Delta\mu) - m_1 \Delta k \Delta\mu/\overline{v}.$$

In the plane x_1, x_2, Eq. (97) represents the hyperbola (see Fig. 2 illustrating the well ordered case)

$$(x_1 - \overline{\alpha}_1)^2 - x_2^2 = \overline{\alpha}_2^2 . \tag{98}$$

Figure 2

With points of its right branch we associate the combination $(74)_+ - (92)_+$ of neighbors, with points of the left branch – combination of $(74)_+ - (92)_-$. We will assume below that $\xi_1 \eta_1 \leq 0$, then

$$\max\{\phi_{1+}, \phi_{3+}, \phi_{3-}\} = \begin{cases} \phi_{3-} & \text{if } (x_1, x_2) \in \Sigma_3, \\ \phi_{1+} & \text{if } (x_1, x_2) \in \Sigma_1, \\ \phi_{3+} & \text{if } (x_1, x_2) \in \Sigma_2. \end{cases}$$

In this classification we should assume that $x_1 = \sigma + \zeta \geq 0$ since the range $(74)_+$ is then selected by (94) for $\xi_1 \eta_1 \leq 0$.

The possibility of contact between $(74)_+$ and $(87)_\pm$ should now be examined. The corresponding analysis reproduces the one preceding Eq. (98); the obvious modification requires that $m_1(m_2)$ should be substituted for $m_2(m_1)$, and $\mu_1(\mu_2)$, $k_1(k_2)$ for $\mu_2(\mu_1)$, $k_2(k_1)$, respectively. Also $-u$ should appear instead of $\bar{u}$, and $-v$ instead of $\bar{v}$.

Equation (98) is then substituted for

$$(x_1 - \alpha_1)^2 - x_2^2 = \alpha_2^2 \tag{99}$$

where

$$\alpha_1 = -u/(m_2 \Delta k \Delta \mu) - m_2 \Delta k \Delta \mu / v, \quad \alpha_2 = -u/(m_2 \Delta k \Delta \mu) + m_2 \Delta k \Delta \mu / v.$$

The hyperbola (99) is reproduced on Figure 3. With points of its right branch we associate the combination $(74)_+ - (87)_-$ of neighbors whereas points of its left branch correspond to the combination $(74)_+ - (87)_+$.

Figure 3

For points belonging to the half–plane $x_1 \geq 0$,

$$\max(\phi_{1+}, \phi_{2-}) = \begin{cases} \phi_{1+} & \text{if } (x_1, x_2) \in F_1, \\ \phi_{2-} & \text{if } (x_1, x_2) \in F_2. \end{cases}$$

If we lay Fig. 3 upon Fig. 2, then, depending on the relationship between u/v and $\bar{u}/\bar{v}$, there will

appear three possibilities. If $u/v > \bar{u}/\bar{v}$, then the half–plane $x_1 \geq 0$ will be partitioned into zones with regimes $(74)_+$, $(87)_-$, $(92)_\pm$ shown on Fig. 4.

Figure 4

If $u/v = \bar{u}/\bar{v}$, then the partitioning is illustrated by Fig. 5; the case $u/v < \bar{u}/\bar{v}$ is given by Fig. 6. In this case, regimes $(87)_-$ and $(92)_-$ are separated by a curve along which $\phi_{2-} = \phi_{3-}$; this curve intersects the x_2–axis at the points

$$x_2 = \pm 2\sqrt{(k_1+\mu_2)/(k_2+\mu_1)}\sqrt{u/v}\,.$$

Figure 5

Figure 6

The case $\xi_1\eta_1 \geq 0$ as well as the "badly ordered" case (ii) can be handled in a similar way.

Once this classification is completed, the final operation of maximizing ϕ with respect to m_1 can be applied to construct the desired material pattern. Obviously, the results described above apply in more general situations, i.e. those when the relative amounts of original materials are prescribed.

Appendix: computation of $\langle S^{-1}\rangle^{-1}$

This procedure is similar to that applied to compute $\langle S\rangle^{-1} = [\langle\Delta\rangle - \omega\times E]^{-1}$ (c.f. Eq. (30)). We start with the expression

$$\langle S^{-1}\rangle = m_1 S_1^{-1} + m_2 S_2^{-1} = m_1\delta_1 + m_2\delta_2 + (m_1\Omega_1 + m_2\Omega_2)\times E$$

where $\delta_1(\delta_2)$ and $\Omega_1(\Omega_2)$ are defined, respectively, by Eqs. (31) and (32) in which we apply $\Delta_1(\Delta_2)$ instead of $\langle\Delta\rangle$.

Using the notation

$$\langle\delta\rangle = m_1\delta_1 + m_2\delta_2,$$

$$\langle\Omega\rangle = m_1\Omega_1 + m_2\Omega_2,$$

we may now invert the matrix

$$\langle S^{-1}\rangle = \langle\delta\rangle + \langle\Omega\rangle\times E.$$

Referring to Eq. (20), we get

$$\langle S^{-1}\rangle^{-1} = [1/(\det\langle\delta\rangle + \langle\Omega\rangle\cdot\cdot\langle\delta\rangle\cdot\cdot\langle\Omega\rangle)]\{\det\langle\delta\rangle\langle\delta\rangle^{-1} + \langle\Omega\rangle\langle\Omega\rangle - (\langle\Omega\rangle\cdot\cdot\langle\delta\rangle)\times E\}$$

where

$$\langle\delta\rangle = m_1\delta_1 + m_2\delta_2 = [m_1/(\det\Delta_1 + \omega\cdot\cdot\Delta_1\cdot\cdot\omega)](\det\Delta_1\cdot\Delta_1^{-1} + \omega\omega) +$$

$$+ [m_2/(\det\Delta_2 + \omega\cdot\cdot\Delta_2\cdot\cdot\omega)](\det\Delta_2\cdot\Delta_2^{-1} + \omega\omega) = \left\langle[\det\Delta/(\det\Delta + \omega\cdot\cdot\Delta\cdot\cdot\omega)]\Delta^{-1}\right\rangle +$$

$$+ \langle 1/(\det\Delta+\omega\cdot\cdot\Delta\cdot\cdot\omega)\rangle\,\omega\omega = \Phi + g\omega\omega,$$

$$\langle\Omega\rangle = m_1\Omega_1 + m_2\Omega_2 = [m_1/(\det\Delta_1+\omega\cdot\cdot\Delta_1\cdot\cdot\omega)]\omega\cdot\cdot\Delta_1 + [m_2/(\det\Delta_2+\omega\cdot\cdot\Delta_2\cdot\cdot\omega)]\omega\cdot\cdot\Delta_2 =$$

$$= \omega\cdot\cdot\langle\Delta/(\det\Delta+\omega\cdot\cdot\Delta\cdot\cdot\omega)\rangle.$$

The matrix $\langle\delta\rangle = \Phi + g\omega\omega$ allows invertion:

$$\langle\delta\rangle^{-1} = (\Phi + g\omega\omega)^{-1} = \Phi^{-1} - [g/(1+g\omega\cdot\cdot\Phi^{-1}\cdot\cdot\omega)](\Phi^{-1}\cdot\cdot\omega)(\Phi^{-1}\cdot\cdot\omega).$$

We also compute $\det\langle\delta\rangle$:

$$\det\langle\delta\rangle = \det(\Phi+g\omega\omega) = (\det\Phi)[1 + g(\omega\cdot\cdot\Phi^{-1}\cdot\cdot\omega)].$$

The final expression for $\langle S^{-1}\rangle^{-1}$ becomes

$$\langle S^{-1}\rangle^{-1} = (1/A)\{\det\Phi[1+g(\omega\cdot\cdot\Phi^{-1}\cdot\cdot\omega)]\Phi^{-1} - g\det\Phi(\Phi^{-1}\cdot\cdot\omega)(\Phi^{-1}\cdot\cdot\omega) +$$

$$\langle\Omega\rangle\langle\Omega\rangle - [\langle\Omega\rangle\cdot\cdot(\Phi + g\omega\omega)] \times E\},$$

where

$$A = \det\Phi[1+g(\omega\cdot\cdot\Phi^{-1}\cdot\cdot\omega)] + \langle\Omega\rangle\cdot\cdot(\Phi+g\omega\omega)\cdot\cdot\langle\Omega\rangle$$

and matrices Φ, $\langle\Omega\rangle$ and the scalar parameter g are defined by the formulas

$$\Phi = \langle [\det\Delta/(\det\Delta + \omega \cdot\cdot \Delta \cdot\cdot \omega)]\Delta^{-1} \rangle,$$

$$\langle \Omega \rangle = \omega \cdot\cdot \langle \Delta/(\det\Delta + \omega \cdot\cdot \Delta \cdot\cdot \omega) \rangle,$$

$$g = \langle 1/(\det\Delta + \omega \cdot\cdot \Delta \cdot\cdot \omega) \rangle.$$

References

1. LURIE, K.A., The Extension of Optimization Problems Containing Controls in the Coefficients, Proceeding of the Royal Society of Edinburg, Vol. 114A, pp. 81–97, 1990.

2. LURIE, K.A., FEDOROV, A.V., and CHERKAEV, A.V., Regularization of Optimal Design Problems for Bars and Plates, Parts 1 and 2, Journal of Optimization Theory and Applications, Vol. 37, pp. 499–521, 1982 and Vol. 37, pp. 523–543, 1982.

3. LURIE, K.A., and CHERKAEV, A.V., The Effective Characteristics of Composite Materials and Optimal Design of Constructions (in Russian), Advances in Mechanics (Poland), Vol. 9, No. 2, pp. 3–81, 1986.

4. BALL, J.M., Convexity Conditions and Existence Theorems in Nonlinear Elasticity, Archive for Rational Mechanics and Analysis, Vol. 63, pp. 337–403, 1977.

5. STRANG, G., The Polyconvexification of F(∇u), Research Report CMA–R09–83, Australian National University, 1983.

6. KOHN, R.V., and STRANG, G., Optimal Design and Relaxation of Variational Problems, Parts 1,2,3, Communications on Pure and Applied Mathematics, Vol. 39, pp. 113–137, 1986; Vol. 39, pp. 139–182, 1986; and Vol. 39, pp. 353–377, 1986.

7. LURIE, A.I., Nonlinear Theory of Elasticity, North Holland, Amsterdam, New York, Oxford, Tokyo, 1990.

8. GIBIANSKII, L.V., and CHERKAEV, A.V., Design of Composite Plates of Extremal Stiffness (in Russian), A.F. Ioffe Institute Report 914, Leningrad, 1984.

9. LURIE, K.A., and LIPTON, R., Direct Solution of an Optimal Layout Problem for Isotropic Heat Conductors in Three Dimensions, Theoretical Aspects of Industrial Design, Edited by David A. Field and Vadim Komkov, SIAM, Philadelphia, pp. 1–11, 1992.

List of Captions

Fig. 1 Rank 2 laminates

Fig. 2 Contact between ranges $(74)_+$ amd $(92)_\pm$: well ordered case

Fig. 3 Contact between ranges $(74)_+$ and $(87)_\pm$: well ordered case

Fig. 4 Partitioning of the half–plane $x_1 \geq 0$; case $u/v > \overline{u}/\overline{v}$

Fig. 5 Partitioning of the half–plane $x_1 \geq 0$; case $u/v = \overline{u}/\overline{v}$

Fig. 6 Partitioning of the half–plane $x_1 \geq 0$; case $u/v < \overline{u}/\overline{v}$

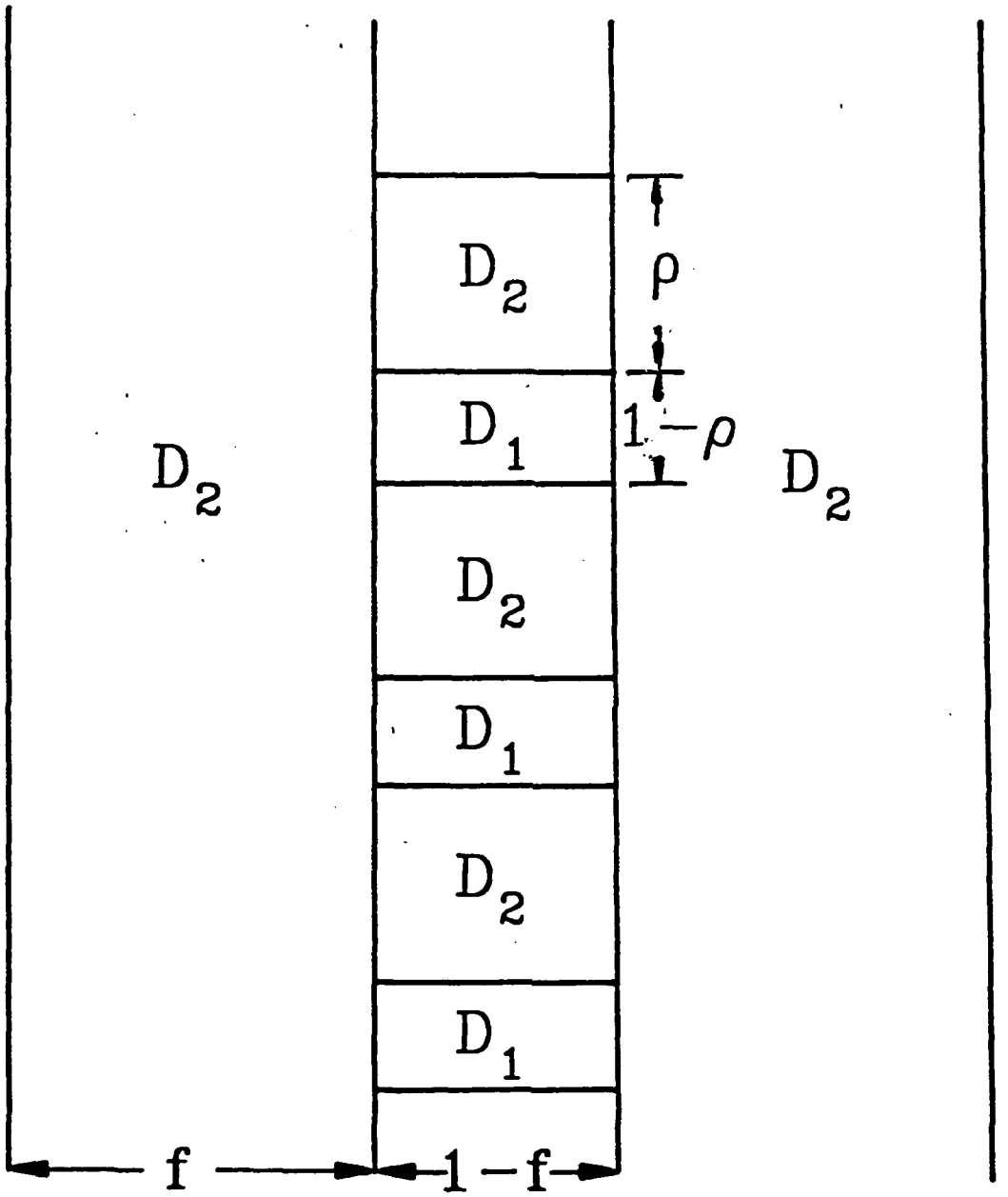


Fig. 1 Rank 2 laminate

K. Lurie Fig 1

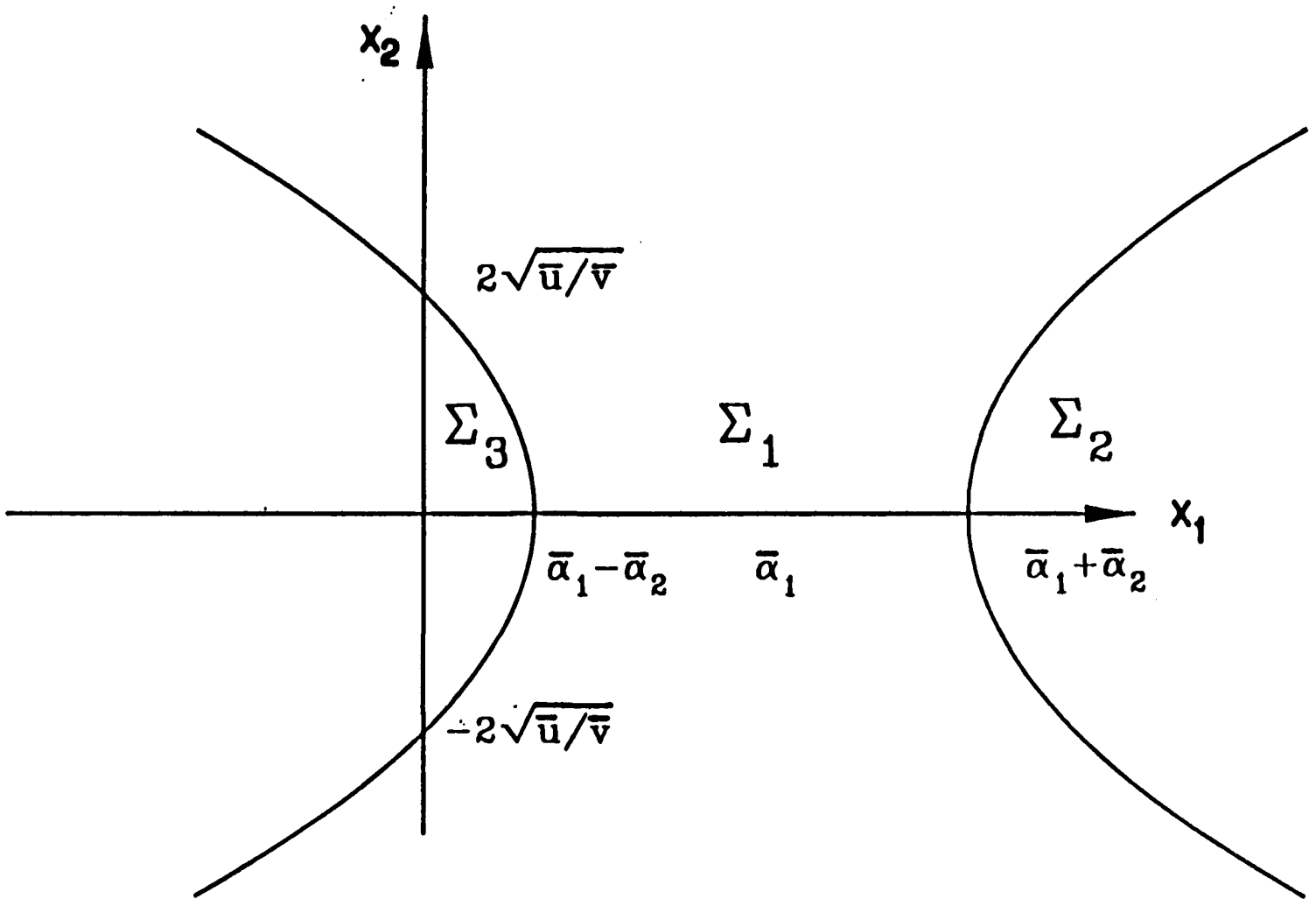

x_2
$2\sqrt{\bar{u}/\bar{v}}$
Σ_3
Σ_1
Σ_2
x_1
$\bar{\alpha}_1-\bar{\alpha}_2$
$\bar{\alpha}_1$
$\bar{\alpha}_1+\bar{\alpha}_2$
$-2\sqrt{\bar{u}/\bar{v}}$

Fig. 2

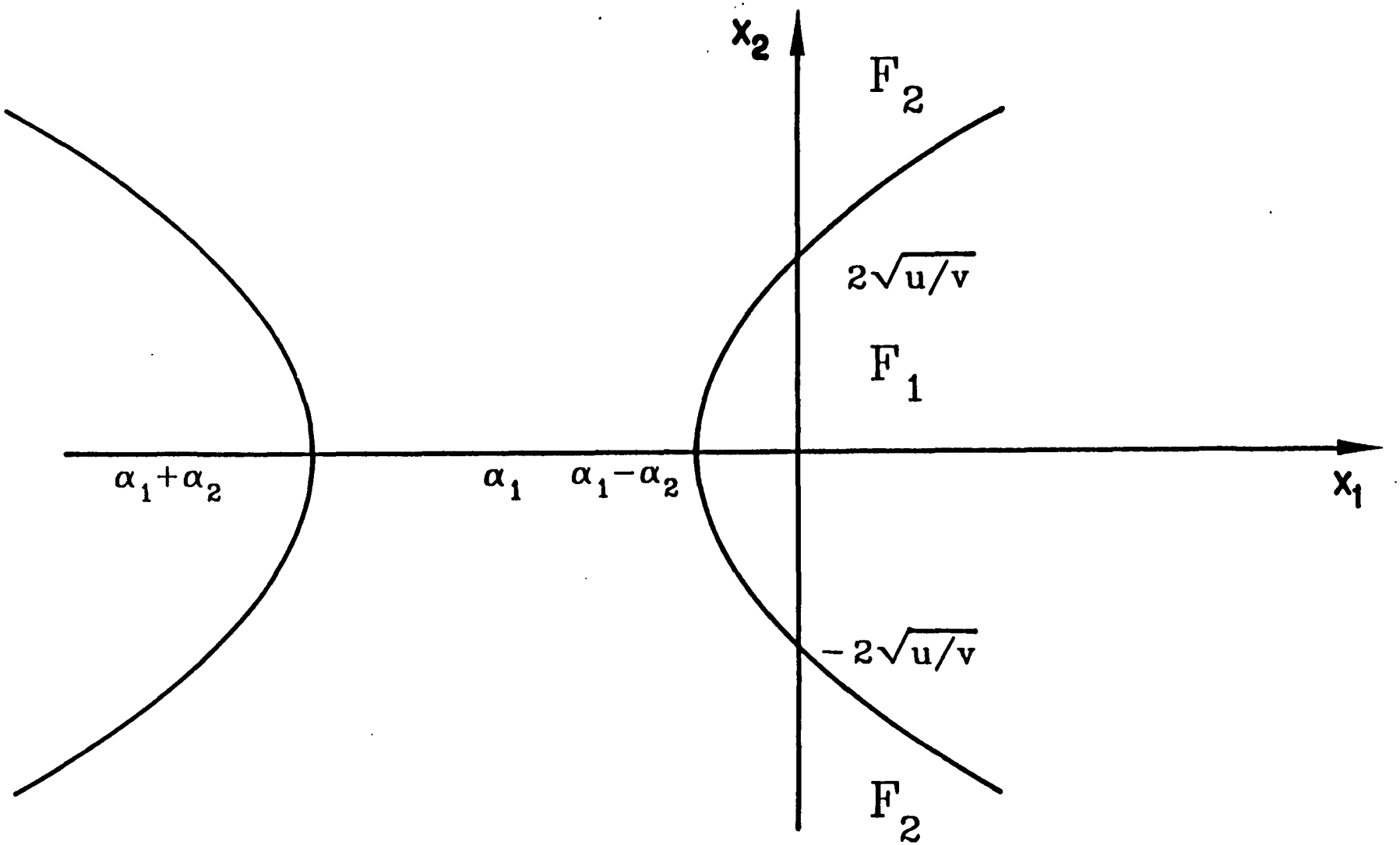

x_2
F_2
$2\sqrt{u/v}$
F_1
$\alpha_1+\alpha_2$
α_1
$\alpha_1-\alpha_2$
x_1
$-2\sqrt{u/v}$
F_2

Fig. 3

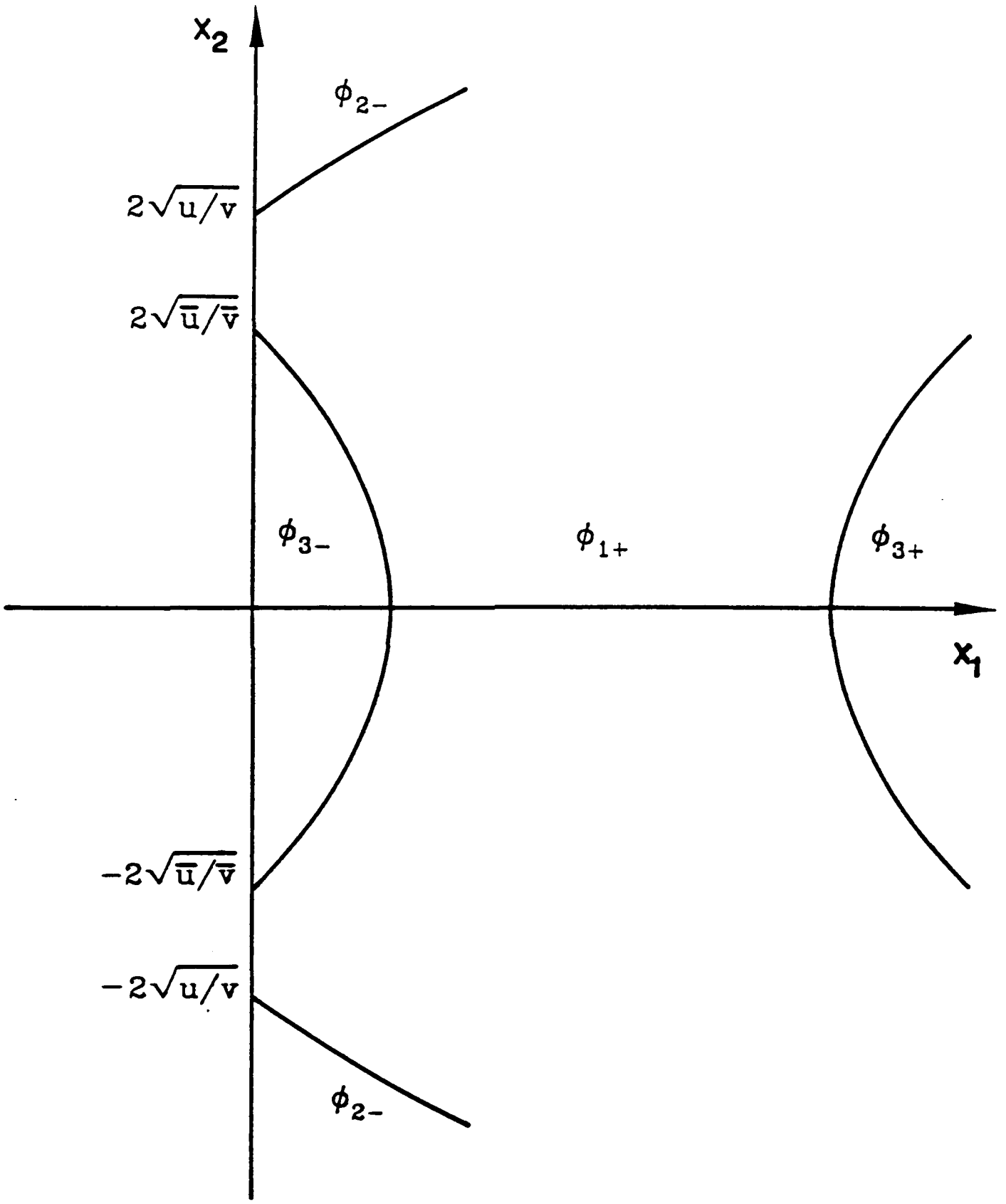

x_2
ϕ_{2-}
$2\sqrt{u/v}$
$2\sqrt{\bar{u}/\bar{v}}$
ϕ_{3-}
ϕ_{1+}
ϕ_{3+}
x_1
$-2\sqrt{\bar{u}/\bar{v}}$
$-2\sqrt{u/v}$
ϕ_{2-}

Fig 4

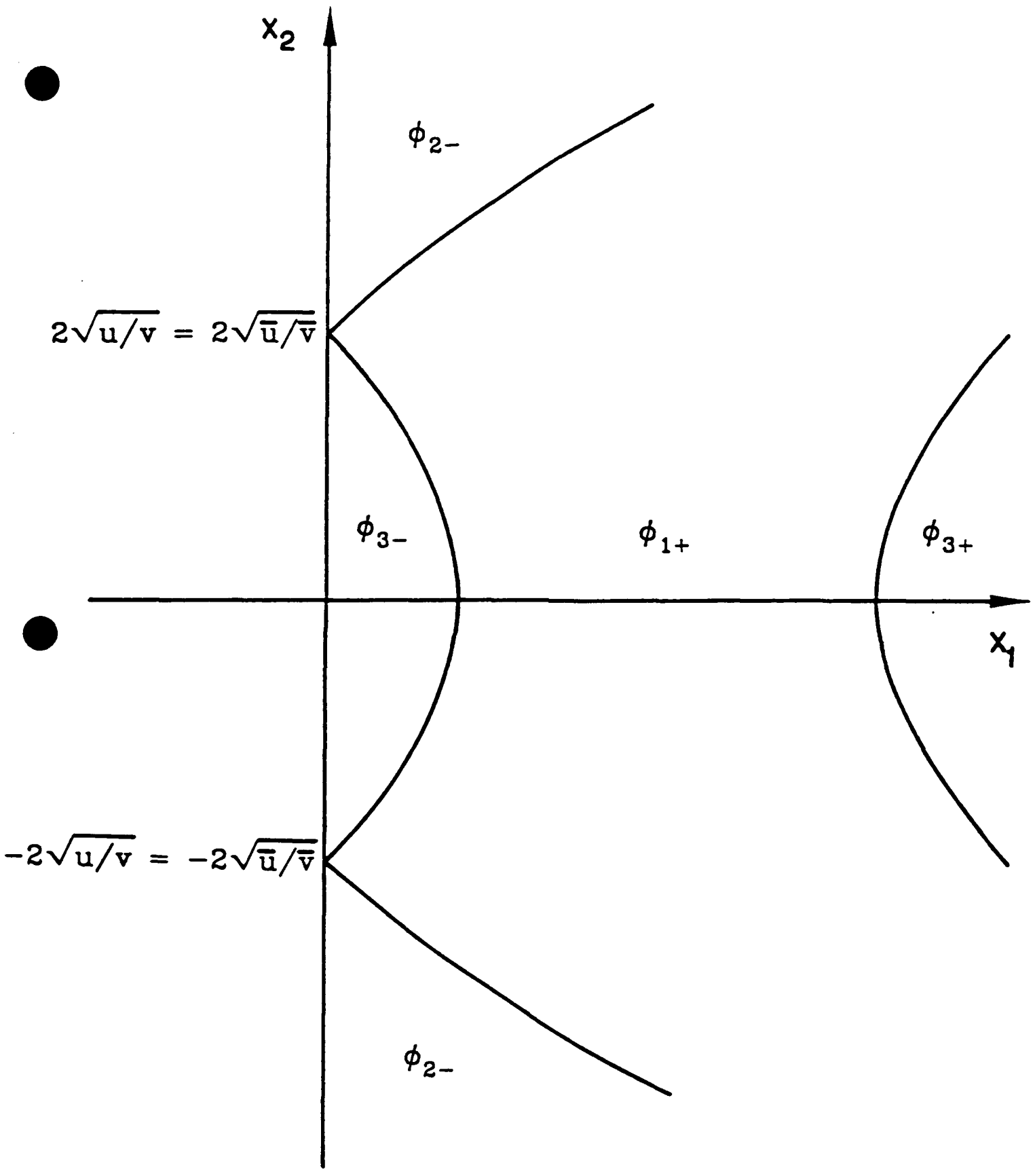

x_2
ϕ_{2-}
$2\sqrt{u/v} = 2\sqrt{\bar{u}/\bar{v}}$
ϕ_{3-}
ϕ_{1+}
ϕ_{3+}
x_1
$-2\sqrt{u/v} = -2\sqrt{\bar{u}/\bar{v}}$
ϕ_{2-}

Fig 5

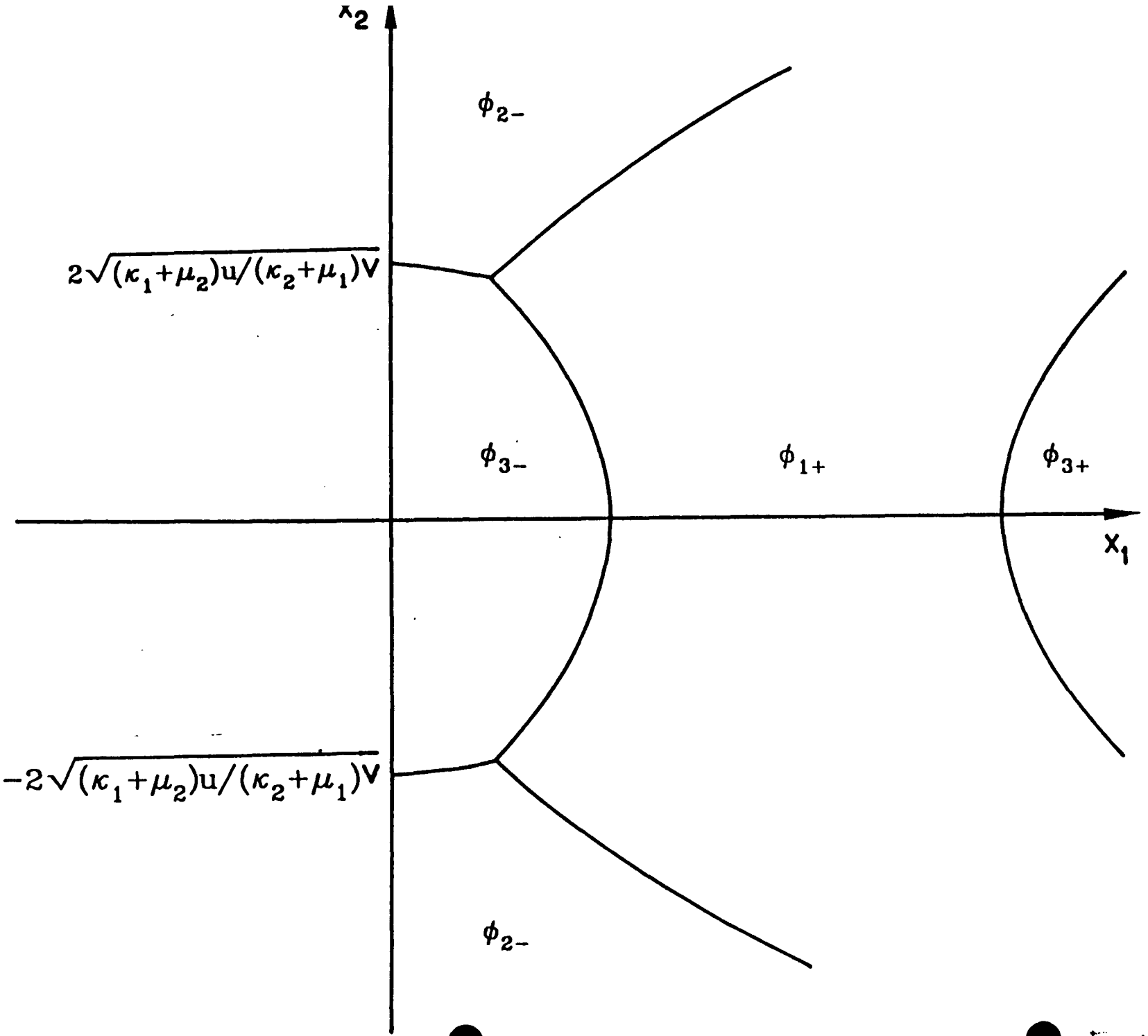

x_2
ϕ_{2-}
$2\sqrt{(\kappa_1+\mu_2)u/(\kappa_2+\mu_1)v}$
ϕ_{3-}
ϕ_{1+}
ϕ_{3+}
x_1
$-2\sqrt{(\kappa_1+\mu_2)u/(\kappa_2+\mu_1)v}$
ϕ_{2-}

Fig 6

Invariant properties of the stress in plane elasticity and equivalence classes of composites

By Andrei V. Cherkaev[1]†, Konstantin A. Lurie[2] and Graeme W. Milton[1]

[1] *Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, New York 10012, U.S.A.*

[2] *Department of Mathematics, Worcester Polytechnic Institute, 100 Institute Road, Worcester, Massachusetts 01609, U.S.A.*

Attention is drawn to the invariance of the stress field in a two-dimensional body loaded at the boundary by fixed forces when the compliance tensor $\mathscr{S}(\boldsymbol{x})$ is shifted uniformly by $\mathscr{S}^{\mathrm{I}}(\lambda, -\lambda)$, where λ is an arbitrary constant and $\mathscr{S}^{\mathrm{I}}(\kappa, \mu)$ is the compliance tensor of a isotropic material with two-dimensional bulk and shear moduli κ and μ. This invariance is explained from two simple observations: first, that in two dimensions the tensor $\mathscr{S}^{\mathrm{I}}(\frac{1}{2}, -\frac{1}{2})$ acts to locally rotate the stress by 90° and the second that this rotated field is the symmetrized gradient of a vector field and therefore can be treated as a strain. For composite materials the invariance of the stress field implies that the effective compliance tensor $\mathscr{S}^*$ also gets shifted by $\mathscr{S}^{\mathrm{I}}(\lambda, -\lambda)$ when the constituent moduli are each shifted by $\mathscr{S}^{\mathrm{I}}(\lambda, -\lambda)$. This imposes constraints on the functional dependence of $\mathscr{S}^*$ on the material moduli of the components. Applied to an isotropic composite of two isotropic components it implies that when the inverse bulk modulus is shifted by the constant $1/\lambda$ and the inverse shear modulus is shifted by $-1/\lambda$, then the inverse effective bulk and shear moduli undergo precisely the same shifts. In particular it explains why the effective Young's modulus of a two-dimensional media with holes does not depend on the Poisson's ratio of the matrix material.

1. Introduction

The purpose of this paper is first to review some results for two-dimensional elasticity, which are not widely known but which may have wide application, and second to give an appropriate physical interpretation of the formal mathematical statements embodied in these results. These results concern the invariance of the stress field in materials with different elastic moduli subject to the same loadings. In fact this property was noted many times in different contexts, beginning with the work of Michell (1899), who studied the behaviour of media with holes. Dundurs (1967) used the complex representation of the Airy stress potential to show the invariance of the stress field for mixtures of two isotropic components. Lurie & Cherkaev (1984*a*) studied the problem in the context of Kirchhoff plate theory which involves the same equations as two-dimensional elasticity. The interest in this problem increased when Day *et al.* (1991) observed, through numerical simulation, that the effective Young's modulus of a two-dimensional media with holes does not

† Present address: Department of Mathematics, University of Utah, Salt Lake City, Utah 84112, U.S.A.

Proc. R. Soc. Lond. A (1992) **438**, 519–529
Printed in Great Britain

depend on the Poisson ratio of the matrix material. Most of these results, together with some new applications, are reviewed by Thorpe & Jasiuk (1992).

In §2 we fix notations and state the standard equations of two-dimensional elasticity (see, for example, Atkin & Fox (1990) for a general reference). In §3 we explain how the material constants can be modified without altering the stress field, and in §4 we discuss the implications for composite materials.

We use bold face small letters to denote vectors. Second-order tensors are denoted by either bold face capital letters or bold face Greek letters, and fourth-order tensors are denoted by caligraphic letters.

2. Equations of planar elasticity

Here we consider a planar simply connected domain Ω filled by an elastic material and loaded on the boundary $\partial\Omega$ by a force $\boldsymbol{t}(\boldsymbol{x})$: this constrains the normal stress $\boldsymbol{\sigma}(\boldsymbol{x})\cdot\boldsymbol{n}$ at $\partial\Omega$ according to

$$\boldsymbol{\sigma}(\boldsymbol{x})\cdot\boldsymbol{n} = \boldsymbol{t}(\boldsymbol{x}), \quad \forall \boldsymbol{x}\in\partial\Omega, \tag{1}$$

where $\boldsymbol{n}$ denotes the normal to the surface. For the body to be in equilibrium the net force and net torque acting on it must be zero:

$$\int_{\partial\Omega} \boldsymbol{t}(\boldsymbol{x}) = 0, \quad \int_{\partial\Omega} \boldsymbol{x}\times\boldsymbol{t}(\boldsymbol{x}) = 0. \tag{2}$$

The equilibrium equation of plane elasticity

$$\nabla\cdot\boldsymbol{\sigma} = 0, \quad \boldsymbol{\sigma} = \boldsymbol{\sigma}^{\mathrm{T}}, \tag{3}$$

may be satisfied identically by means of the Airy potential function ϕ (Atkin & Fox 1990) in terms of which the stress is given by

$$\boldsymbol{\sigma} = \begin{pmatrix} \phi_{,22} & -\phi_{,21} \\ -\phi_{,12} & \phi_{,11} \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} \phi_{,11} & \phi_{,12} \\ \phi_{,12} & \phi_{,22} \end{pmatrix}\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \tag{4}$$

where the comma means differentiation with respect to subsequent cartesian directions. By introducing the tensor of rotation by a right angle,

$$\boldsymbol{R} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \tag{5}$$

and the differential operator,

$$\nabla\nabla = \begin{pmatrix} \partial^2/\partial x_1\partial x_1 & \partial^2/\partial x_1\partial x_2 \\ \partial^2/\partial x_2\partial x_1 & \partial^2/\partial x_2\partial x_2 \end{pmatrix}, \tag{6}$$

we can re-express (4) in the more compact form

$$\boldsymbol{\sigma} = \boldsymbol{R}^{\mathrm{T}}\cdot(\nabla\nabla\phi)\cdot\boldsymbol{R}. \tag{7}$$

Note that $\boldsymbol{R}$ is the two-dimensional counterpart of the completely antisymmetric Levi-Civita tensor. We write it as $\boldsymbol{R}$ to emphasize the key point (made in §3) that the stress field $\boldsymbol{\sigma}$ rotated locally by 90° can be treated as a strain field because $\nabla\nabla\phi$ is the symmetrized gradient of $\nabla\phi$.

The requirement that Ω be simply connected is needed to ensure that ϕ is single valued. Multiply connected domains also have a single valued ϕ provided that

integrals of the type (2) vanish separately on each internal boundary. Under this restriction our analysis applies equally well to multiply connected domains.

Now given any symmetric 2×2 matrix $\boldsymbol{A}$ with elements a_{ij} we have

$$\boldsymbol{R}^{\mathrm{T}}\cdot\boldsymbol{A}\cdot\boldsymbol{R}=\begin{pmatrix} a_{22} & -a_{12}\\ -a_{12} & a_{11}\end{pmatrix}=\begin{pmatrix} a_{11}+a_{22} & 0\\ 0 & a_{11}+a_{22}\end{pmatrix}-\begin{pmatrix} a_{11} & a_{12}\\ a_{12} & a_{22}\end{pmatrix}=\boldsymbol{I}(\mathrm{Tr}\,\boldsymbol{A})-\boldsymbol{A}. \quad (8)$$

where Tr denotes the trace of the matrix and $\boldsymbol{I}$ denotes the identity tensor. Consequently we can also rewrite (7) using a fourth-order tensor $\mathscr{R}$, representing the action of rotation by a right angle on symmetric second-order tensor, defined as follows:

$$\mathscr{R}_{ijkl}=\tfrac{1}{2}\delta_{ij}\delta_{kl}-\tfrac{1}{2}(\delta_{ik}\delta_{jl}+\delta_{il}\delta_{jk}-\delta_{ij}\delta_{kl}). \quad (9)$$

in the form

$$\boldsymbol{\sigma}=\mathscr{R}:\nabla\nabla\phi. \quad (10)$$

Here, as elsewhere, the symbol : denotes a double contraction of indices in the same way that the symbol $\cdot$ denotes a single contraction. Specifically, if $\mathscr{A}$ and $\mathscr{B}$ are fourth-order tensors with cartesian elements $\mathscr{A}_{ijkl}$ and $\mathscr{B}_{ijkl}$, and $\boldsymbol{A}$ and $\boldsymbol{B}$ are second-order tensors with cartesian elements A_{ij} and B_{ij} then we define

$$\left.\begin{aligned}(\mathscr{A}:\mathscr{B})_{ijkl}&=\sum_{m=1}^{2}\sum_{n=1}^{2}\mathscr{A}_{ijmn}\mathscr{B}_{mnkl},\\ (\mathscr{A}:\boldsymbol{B})_{ij}&=\sum_{m=1}^{2}\sum_{n=1}^{2}\mathscr{A}_{ijmn}B_{mn},\\ \boldsymbol{A}:\boldsymbol{B}&=\sum_{m=1}^{2}\sum_{n=1}^{2}A_{mn}B_{mn}.\end{aligned}\right\} \quad (11)$$

In other words $\mathscr{A}:\boldsymbol{B}$ is the second-order tensor that results when $\mathscr{A}$ acts upon $\boldsymbol{B}$, $\mathscr{A}:\mathscr{B}$ is a fourth-order tensor and represents the product of the two tensors $\mathscr{A}$ and $\mathscr{B}$, while $\boldsymbol{A}:\boldsymbol{B}$ represents the trace of the second-order tensor $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{B}$, or equivalently the inner product of $\boldsymbol{A}$ and $\boldsymbol{B}$.

The equilibrium condition for the stress is to be supplemented by the constitutive (Hooke's) law

$$\boldsymbol{\varepsilon}=\mathscr{S}:\boldsymbol{\sigma}, \quad (12)$$

where $\boldsymbol{\varepsilon}$ is the strain tensor and $\mathscr{S}$ denotes the fourth-order compliance tensor of the elastic material possibly depending on the position $\boldsymbol{x}$.

If the material is isotropic, then the elasticity tensor $\mathscr{C}=(\mathscr{S})^{-1}=\mathscr{C}^{\mathrm{I}}(\kappa,\mu)$ can be expressed in the form

$$\mathscr{C}^{\mathrm{I}}_{ijkl}(\kappa,\mu)=\kappa\delta_{ij}\delta_{kl}+\mu(\delta_{ik}\delta_{jl}+\delta_{il}\delta_{jk}-\delta_{ij}\delta_{kl}), \quad (13)$$

and its associated compliance tensor $\mathscr{S}=\mathscr{S}^{I}(\kappa,\mu)$ is given by

$$\mathscr{S}^{\mathrm{I}}_{ijkl}(\kappa,\mu)=(1/4\kappa)\,\delta_{ij}\delta_{kl}+(1/4\mu)\,(\delta_{ik}\delta_{jl}+\delta_{il}\delta_{jk}-\delta_{ij}\delta_{kl}). \quad (14)$$

where κ denotes the two-dimensional bulk modulus and μ denotes the shear modulus. It should be noted that κ is not the same as the customarily used three-dimensional bulk modulus k: however, for the problem of generalized plane stress where a thin plate of uniform thickness is deformed in its own plane they are related by the formula

$$\kappa=9k\mu/(3k+4\mu). \quad (15)$$

In this context the fields $\boldsymbol{\varepsilon}$ and $\boldsymbol{\sigma}$ represent the transverse components of the strain and stress fields averaged over the thickness of the plate. (For more explanation see, for example, Atkin & Fox (1990).)

The strain field $\boldsymbol{\varepsilon}(\boldsymbol{x})$ is connected with the displacement $\boldsymbol{u}(\boldsymbol{x})$ via

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^{\mathrm{T}}). \tag{16}$$

The last equation is equivalent to requiring that the strain satisfy the scalar differential constraint

$$\nabla \cdot [\nabla \cdot (\boldsymbol{R}^{\mathrm{T}} \cdot \boldsymbol{\varepsilon} \cdot \boldsymbol{R})] = 0, \tag{17}$$

or, in the notation of (6) and (9),

$$\nabla\nabla : \mathscr{R} : \boldsymbol{\varepsilon} = 0. \tag{18}$$

Indeed, we have

$$\boldsymbol{R}^{\mathrm{T}} \cdot \boldsymbol{\varepsilon} \cdot \boldsymbol{R} = \begin{pmatrix} \epsilon_{22} & -\epsilon_{12} \\ -\epsilon_{12} & \epsilon_{11} \end{pmatrix} = \begin{pmatrix} u_{2,2} & -\frac{1}{2}(u_{1,2}+u_{2,1}) \\ -\frac{1}{2}(u_{1,2}+u_{2,1}) & u_{1,1} \end{pmatrix} \tag{19}$$

and so one can easily check that

$$\nabla\nabla : \mathscr{R} : \boldsymbol{\varepsilon} = \nabla \cdot [\nabla \cdot (\boldsymbol{R}^{\mathrm{T}} \cdot \boldsymbol{\varepsilon} \cdot \boldsymbol{R})] = (u_{2,2})_{,11} - (u_{1,2}+u_{2,1})_{,12} + (u_{1,1})_{,22} = 0. \tag{20}$$

Note that (18) is the well-known infinitesimal strain compatibility condition of two-dimensional linear elasticity. Combining the equations (10), (12), and (18) we arrive at the fourth-order differential equation for the Airy potential

$$\nabla\nabla : \mathscr{S}_{\perp} : \nabla\nabla\phi = 0, \tag{21}$$

where $\mathscr{S}_{\perp}$ is $\mathscr{S}$ rotated by a right angle:

$$\mathscr{S}_{\perp} = \mathscr{R} : \mathscr{S} : \mathscr{R}. \tag{22}$$

Clearly, the isotropic tensor $\mathscr{S}^I$ coincides with the rotated one:

$$\mathscr{S}^I(\kappa, \mu) = \mathscr{S}^I_{\perp}(\kappa, \mu), \tag{23}$$

and equation (21) takes the form

$$\begin{aligned}\Big[&\Big(\frac{\partial^2}{\partial x_1 \partial x_1} + \frac{\partial^2}{\partial x_2 \partial x_2}\Big)\Big(\frac{1}{\kappa} + \frac{1}{\mu}\Big)\Big(\frac{\partial^2}{\partial x_1 \partial x_1} + \frac{\partial^2}{\partial x_2 \partial x_2}\Big) \\ &- \Big(\frac{\partial^2}{\partial x_1 \partial x_1}\Big)\frac{2}{\mu}\Big(\frac{\partial^2}{\partial x_2 \partial x_2}\Big) - \Big(\frac{\partial^2}{\partial x_2 \partial x_2}\Big)\frac{2}{\mu}\Big(\frac{\partial^2}{\partial x_1 \partial x_1}\Big) + \Big(\frac{\partial^2}{\partial x_1 \partial x_2}\Big)\frac{4}{\mu}\Big(\frac{\partial^2}{\partial x_1 \partial x_2}\Big)\Big]\phi = 0. \end{aligned} \tag{24}$$

We remark in passing that (21) also describes the bending of thin plates according to Kirchoff theory: ϕ represents the vertical deflection of a horizontal plate. $\nabla\nabla\phi$ is the tensor of curvature of the plate. $\mathscr{S}_{\perp}$ is the tensor of flexural rigidity dependent on the local thickness of the plate and $\mathscr{S}_{\perp} : \nabla\nabla\phi$ represents the tensor of bending moments, satisfying the equilibrium equation (21) (Timoshenko 1959). Thus all the ensuing analysis applies equally well to the plate equation (see Lurie & Cherkaev 1984*a*).

3. Equivalent plane elasticity problems

It is remarkable that any potential ϕ satisfying the equation (21) is for any choice of the parameter λ also a solution of

$$\nabla\nabla : \mathscr{S}'_{\perp} : \nabla\nabla\phi = 0, \tag{25}$$

where

$$\mathscr{S}'_{\perp} = \mathscr{R} : \mathscr{S}' : \mathscr{R}, \quad \mathscr{S}' = \mathscr{S} + \mathscr{S}^I(\lambda, -\lambda) \tag{26}$$

implying that the Airy potential and therefore the stress field remains unchanged when the material constants are modified from $\mathscr{S}$ to $\mathscr{S}'$. We will call such a pair of materials with compliance tensors $\mathscr{S}(\boldsymbol{x})$ and $\mathscr{S}'(\boldsymbol{x})$ equivalent. The result (25) can be established directly by substituting $\kappa = \lambda, \mu = -\lambda$ in (24) thereby showing that

$$\nabla\nabla : \mathscr{S}^{\mathrm{I}}(\lambda, -\lambda) : \nabla\nabla\phi = 0 \tag{27}$$

for any function ϕ. Formally speaking the compliance tensor $\mathscr{S}^{\mathrm{I}}(\lambda, -\lambda)$ is nothing other than a multiple of the fourth-order rotation tensor $\mathscr{R}$: from (10) and (14) it follows that

$$\mathscr{S}^{\mathrm{I}}(\lambda, -\lambda) = (1/2\lambda)\,\mathscr{R}. \tag{28}$$

The physical explanation for the identity (27) is that the stress field $\boldsymbol{\sigma}$ when rotated by a right angle at each point results in a field (see (7))

$$\boldsymbol{\varepsilon}^0 = \boldsymbol{R}\cdot\boldsymbol{\sigma}\cdot\boldsymbol{R}^{\mathrm{T}} = \mathscr{R} : \boldsymbol{\sigma} = \nabla\nabla\phi \tag{29}$$

satisfying the same differential constraints (17) as a strain field. Indeed, it is clear that

$$\nabla\nabla : \boldsymbol{R}^{\mathrm{T}}\cdot\boldsymbol{\varepsilon}^0\cdot\boldsymbol{R} = \nabla\nabla : \boldsymbol{\sigma} = \nabla\cdot(\nabla\cdot\boldsymbol{\sigma}) = 0. \tag{30}$$

Furthermore, from (29) it follows that

$$\boldsymbol{\varepsilon}^0 = \tfrac{1}{2}(\nabla\boldsymbol{u}^0 + (\nabla\boldsymbol{u}^0)^{\mathrm{T}}), \quad \boldsymbol{u}^0 = \nabla\phi. \tag{31}$$

So this strain field is associated with a special vector displacement field $\boldsymbol{u}^0$ which is in turn the gradient of the Airy potential function. Note, that the displacements $\boldsymbol{u}$ and $\boldsymbol{u}'$ in the equivalent materials $\mathscr{S}$ and $\mathscr{S}'$ under the same surface loading $\boldsymbol{t}(\boldsymbol{x})$ differ by a multiple of $\boldsymbol{u}^0$

$$\boldsymbol{u}' = \boldsymbol{u} + \boldsymbol{u}^0/(2\lambda) = \boldsymbol{u} + \nabla\phi/(2\lambda). \tag{32}$$

In particular this gives a direct way of finding the gradient of the Airy stress potential $\nabla\phi$ from measurements of the displacements $\boldsymbol{u}$ and $\boldsymbol{u}'$.

Another explanation of the equivalence (25), (26) follows from consideration of the elastic energy variational principle

$$\min_{\boldsymbol{\sigma}} \int_{\Omega} W(\mathscr{S}), \tag{33}$$

where the minimum is over all $\boldsymbol{\sigma}$ satisfying (1) and (3), and

$$W(\mathscr{S}, \boldsymbol{\sigma}) = \boldsymbol{\sigma} : \mathscr{S} : \boldsymbol{\sigma} = \nabla\nabla\phi : \mathscr{R} : \mathscr{S} : \mathscr{R} : \nabla\nabla\phi. \tag{34}$$

The Euler–Lagrange equation associated with this minimization over $\boldsymbol{\sigma}$, or equivalently ϕ, coincides with (21). It is easy to observe that the integrand $W(\mathscr{S}^{\mathrm{I}}(\lambda, -\lambda), \boldsymbol{\sigma})$ can be expressed as the divergence of a vector field

$$\begin{aligned} W(\mathscr{S}^{\mathrm{I}}(\lambda, -\lambda), \boldsymbol{\sigma}) &= (1/2\lambda)\,\nabla\nabla\phi : \mathscr{R} : \nabla\nabla\phi \\ &= (\phi_{,11}\phi_{,22} - \phi_{,12}^2)/\lambda = \nabla\cdot\boldsymbol{v}, \end{aligned} \tag{35}$$

where

$$\boldsymbol{v} = \frac{1}{2\lambda}\nabla\phi\cdot\mathscr{R} : \nabla\nabla\phi = \frac{1}{2\lambda}\begin{pmatrix}\phi_{,1}\phi_{,22} - \phi_{,2}\phi_{,12} \\ \phi_{,2}\phi_{,11} - \phi_{,1}\phi_{,12}\end{pmatrix}. \tag{36}$$

and therefore its integral depends only on the boundary terms. Such functions are called null lagrangians (see, for example, Ball *et al.* 1981) because their Euler–

Lagrange equations vanish identically. They play an important role in the theories of quasi-convexity (Kohn & Strang 1986) and compensated compactness (Tartar 1979) used now in many applications including optimal design, homogenization, liquid crystals, and bounds on effective moduli of composite materials.

In fact these arguments parallel results of Dykhne (1970), and Stroud & Berman (1984) (see also Keller 1964; Milton 1988) for conductivity in two dimensions in the presence of a magnetic field perpendicular to the plane of conduction, which generates through the Hall-effect a non-symmetric conductivity tensor. They observed that any electrical potential $V(\boldsymbol{x})$ satisfying the equation

$$\nabla\cdot\boldsymbol{\Sigma}(\boldsymbol{x})\cdot\nabla V=0, \tag{37}$$

where $\boldsymbol{\Sigma}(\boldsymbol{x})$ represents the conductivity of the body, also solves

$$\nabla\cdot(\boldsymbol{\Sigma}(\boldsymbol{x})+\lambda\boldsymbol{R})\cdot\nabla V=0, \tag{38}$$

for any choice of λ. Just as a stress field rotated by a right angle produces a strain field so does an electric field, when rotated by a right angle produce a current (divergence free) field. (Note, however, that a strain field rotated by a right angle does not produce a stress field whereas a current field, when rotated by a right angle does produce an electric field.)

Now we investigate some particular physical consequences that follow from (25). The representation (25) shows that the coefficients of the compliance tensor $\mathscr{S}$ and consequently the strain field $\boldsymbol{\varepsilon}$ cannot be determined from the solution $\boldsymbol{\sigma}$ of the boundary value problem (see (25), (1), and (7)). For example, instead of a locally isotropic elastic material with moduli κ and μ one can substitute into the equation (25) the moduli κ' and μ':

$$\frac{1}{\kappa'(\boldsymbol{x})}=\frac{1}{\kappa(\boldsymbol{x})}+\frac{1}{\lambda};\quad \frac{1}{\mu'(\boldsymbol{x})}=\frac{1}{\mu(\boldsymbol{x})}-\frac{1}{\lambda}. \tag{39}$$

and it does not change the solution ϕ nor the stress field $\boldsymbol{\sigma}$. We will call such a pair of materials equivalent. Note, that the substituted material does not necessarily have positive moduli κ' and μ' and therefore may not have a physical interpretation.

Let us consider the relation between the Young moduli E and E' and the Poisson ratios ν and ν' of equivalent materials. Comparing the constitutive law (12) with the defining equation

$$\epsilon_{11}=(1/E)(\sigma_{11}-\nu\sigma_{22}), \tag{40}$$

for the Young modulus E and the Poisson ratio ν we obtain the relations

$$\frac{1}{E}=\frac{1}{4\kappa}+\frac{1}{4\mu},\quad \nu=\left(\frac{1}{\mu}-\frac{1}{\kappa}\right)\Big/\left(\frac{1}{\kappa}+\frac{1}{\mu}\right)=\frac{\kappa-\mu}{\kappa+\mu}. \tag{41}$$

It is clear from (39) that the equivalent materials have the same Young moduli at each point in the body

$$E'(\boldsymbol{x})=E(\boldsymbol{x}). \tag{42}$$

and Poisson ratios linked by

$$\nu'(\boldsymbol{x})=\nu(\boldsymbol{x})-E(\boldsymbol{x})/2\lambda. \tag{43}$$

where λ is an arbitrary parameter constant throughout the body.

In particular this implies that the stress field in a loaded body with constant Young modulus does not react when the Poisson ratio $\nu(\boldsymbol{x})$ is shifted uniformly.

4. Applications to composites

Finally, there are useful implications of the identity (27) to the theory of composites. Suppose now that the body Ω is a statistically homogeneous (or periodic) composite with grain sizes much smaller than the size of Ω, subject to the loading (1). This composite can be replaced by an equivalent homogeneous effective medium with compliance tensor $\mathscr{S}^*$, connecting the locally averaged stress and strain tensor fields,

$$\langle \boldsymbol{\varepsilon} \rangle = \mathscr{S}^* : \langle \boldsymbol{\sigma} \rangle, \tag{44}$$

where $\langle \cdot \rangle$ denotes local averaging over a test sphere $\Theta(\boldsymbol{x})$ centred at $\boldsymbol{x}$, which is larger than the grain sizes but smaller than all other characteristic lengths such as the size of Ω and the length scale over which $\boldsymbol{t}(\boldsymbol{x})$ varies. To obtain the effective compliance $\mathscr{S}^*$ it suffices to consider an infinite body of the composite loaded uniformly, and the averages $\langle \cdot \rangle$ can then be taken over the whole body, or equivalently over a period cell.

Since the stress $\boldsymbol{\sigma}$ is the same in the equivalent materials $\mathscr{S}(\boldsymbol{x})$ and $\mathscr{S}'(\boldsymbol{x})$ (where $\mathscr{S}'(\boldsymbol{x})$ is given by (26)) when they are loaded uniformly in the same way, it follows that the strain fields satisfy the relation

$$\boldsymbol{\varepsilon}' = \mathscr{S}' : \boldsymbol{\sigma} = \mathscr{S} : \boldsymbol{\sigma} + \mathscr{S}^{\mathrm{I}}(\lambda, -\lambda) : \boldsymbol{\sigma} = \boldsymbol{\varepsilon} + \mathscr{S}^{\mathrm{I}}(\lambda, -\lambda) : \boldsymbol{\sigma}. \tag{45}$$

Taking the average of this relation over a period cell, and remembering that $\mathscr{S}^{\mathrm{I}}(\lambda, -\lambda)$ does not depend on $\boldsymbol{x}$, gives

$$\mathscr{S}'^* : \langle \boldsymbol{\sigma} \rangle = \mathscr{S}^* : \langle \boldsymbol{\sigma} \rangle + \mathscr{S}^{\mathrm{I}}(\lambda, -\lambda) : \langle \boldsymbol{\sigma} \rangle. \tag{46}$$

Since this holds for all uniform loadings, or equivalently for all values of the average stress $\langle \boldsymbol{\sigma} \rangle$, we deduce from (46) that the effective tensors are linked through the equation

$$\mathscr{S}'^* = \mathscr{S}^* + \mathscr{S}^{\mathrm{I}}(\lambda, -\lambda). \tag{47}$$

In other words the effective compliance tensor is translated by the same tensor as the compliance tensor of the initial inhomogeneous medium. Applied to an isotropic composite of isotropic components it implies that when the moduli $1/\kappa(\boldsymbol{x})$ and $1/\mu(\boldsymbol{x})$ are shifted according to (39) their effective moduli are shifted in the same way

$$\frac{1}{\kappa'^*} = \frac{1}{\kappa^*} + \frac{1}{\lambda}; \quad \frac{1}{\mu'^*} = \frac{1}{\mu^*} - \frac{1}{\lambda}. \tag{48}$$

This observation explains the numerical results of Day *et al.* (1991): see also Thorpe & Jasiuk (1991). They found that the relative Young's modulus E^*/E of a two-dimensional sheet containing a statistically isotropic distribution of circular holes, overlapping or not, is independent of the Poisson's ratio ν of the sheet. The reason for this is clear. From dimensional considerations E^*/E can only depend on ν and on the geometrical configuration of the holes. So, without loss of generality, let us suppose the Young's modulus E of the plate remains fixed. A uniform shift in the Poisson's ratio of the sheet from ν to ν', while keeping the Young's modulus $E' = E$, corresponds to a transformation of the form (42) and (43) with $\lambda = E/2(\nu - \nu')$. In other words plates with different Poisson's ratios but sharing the same Young's

modulus and geometrical configuration of holes are equivalent. Under this transformation the holes remain holes and according to (48) and (41) the effective Young's modulus E^* and effective Poisson's ratio ν^* are transformed to

$$E'^* = E^*, \tag{49}$$

$$\nu'^* = \nu^* - (\nu - \nu')\, E^*/E. \tag{50}$$

Consequently the ratio E^*/E remains unchanged, i.e. it can only depend on the geometrical configuration of the holes.

An interesting consequence of (48) is that it leads to a very simple proof of the Hashin–Shtrikman–Hill bound on the bulk modulus κ^* (Hashin & Shtrikman 1963; Hill 1964). We begin with the Voigt bound on κ'^* (Hill 1952)

$$\kappa'^* \leqslant \langle \kappa' \rangle, \tag{51}$$

which holds for all choices of λ such that κ' and μ' given by (39) remain non-negative. In particular, when λ takes its extreme value,

$$\lambda = \mu^+ = \max_x \mu(x), \tag{52}$$

(51) implies

$$\left(\frac{1}{\kappa^*} + \frac{1}{\mu^+}\right)^{-1} \leqslant \left\langle \left(\frac{1}{\kappa(x)} + \frac{1}{\mu^+}\right)^{-1} \right\rangle, \tag{53}$$

which is the Hashin–Shtrikman–Hill bound. This is one of the simplest examples of the so-called translation method to bounding the effective moduli of composites. The method generalizes the idea of equivalence, and it not only provides an alternative derivation of the Hashin–Shtrikman bounds on the effective conductivity, bulk and shear moduli (Lurie & Cherkaev 1984*a*, *b*, 1986*a*; Tartar 1985; Francfort & Murat 1986; Milton 1990, 1991) but also generates coupled estimates on the possible (κ^*, μ^*) pairs (Cherkaev & Gibiansky 1991), exact estimates of the elastic energy stored in the composite for a given applied field (Lurie & Cherkaev 1986*b*; Gibiansky & Cherkaev 1987; Allaire & Kohn 1991), and sharp bounds on the effective elastic moduli of polycrystalline materials (Avellaneda & Milton 1989; Cherkaev *et al.* 1991).

The equivalence (47) was the key point used in the paper by Lurie & Cherkaev (1984*a*) to get the bounds for the set of all possible effective tensors $\mathscr{S}^*(x)$ when $\mathscr{S}(x)$ is isotropic and either the bulk or shear modulus of the material is constant. When the shear modulus is constant then the effective tensor is necessarily isotropic and shares the same shear modulus as the components (Hill 1964). To obtain the effective bulk modulus one can choose $\lambda = \mu$ giving

$$\frac{1}{\kappa'(x)} = \frac{1}{\kappa(x)} + \frac{1}{\mu}, \quad \frac{1}{\mu'} = 0. \tag{54}$$

From (25) it is immediately clear that the Airy potential satisfies

$$\Delta\left(\frac{1}{\kappa(x)} + \frac{1}{\mu}\right)\Delta\phi = 0, \tag{55}$$

where Δ denotes the laplacian, implying that the scalar field

$$\operatorname{Tr}\varepsilon'(x) = \tfrac{1}{2}(1/\kappa(x) + 1/\mu)\,\Delta\phi(x) \tag{56}$$

is harmonic (see (55)).

The physical explanation of this is quite clear. The equivalent material has infinite shear modulus and so its only possible deformations are conformal ones since a

change in the angle between two lines under deformation indicates shear. In particular, if the deformation in the equivalent media is on average uniform then it must necessarily be a uniform dilation. The associated strain field in the equivalent media is independent of $\boldsymbol{x}$, and proportional to the identity tensor

$$\varepsilon'_{ij}(\boldsymbol{x}) = \langle \varepsilon'_{ij} \rangle = \alpha \delta_{ij} \quad \forall \boldsymbol{x}. \tag{57}$$

and the trace of the average stress field is

$$\begin{aligned} \operatorname{Tr} \langle \boldsymbol{\sigma}' \rangle = \langle \Delta \phi(\boldsymbol{x}) \rangle &= 2 \langle (1/\kappa(\boldsymbol{x}) + 1/\mu)^{-1} \operatorname{Tr} \boldsymbol{\varepsilon}' \rangle \\ &= 2 \langle (1/\kappa(\boldsymbol{x}) + 1/\mu)^{-1} \rangle \operatorname{Tr} \langle \boldsymbol{\varepsilon}' \rangle. \end{aligned} \tag{58}$$

This in conjunction with (48) shows that the effective bulk modulus κ^* satisfies

$$\left(\frac{1}{\kappa^*} + \frac{1}{\mu} \right)^{-1} = \left\langle \left(\frac{1}{\kappa(\boldsymbol{x})} + \frac{1}{\mu} \right)^{-1} \right\rangle. \tag{59}$$

independent of the microstructure of the composite. The same conclusion was also reached by Hill (1964), and by Francfort & Tartar (1991), who showed the relation can be generalized to three-dimensional composites with constant shear modulus. The result also follows from the Hashin–Shtrikman bounds (Hashin & Shtrikman 1963) which collapse to the single relation (59) when the shear modulus is constant.

Since the equivalent medium has infinite shear modulus its Poisson ratio given by (41) is -1. Materials with Poisson's ratios close to -1 although not yet found in nature can nevertheless be constructed (see, for example, Lakes 1991; Milton 1992).

In a similar way, supposing that the bulk modulus is constant and choosing $\lambda = -\kappa$ one can get the equivalent incompressible medium. For two-phase composites of isotropic incompressible media, mixed in prescribed proportions, Lipton (1988) has obtained a complete characterization of the set of all possible effective elasticity tensors. This immediately gives, via (47), a complete knowledge of the possible elasticity tensors when the bulk-modulus is finite and constant in both phases.

Clearly equivalent media differ very much in their elastic behaviour. The equivalence means that when subject to the same loading conditions they (as well as infinitely many other equivalent materials) possess the same stress tensor in each point of the body Ω. Finally, note that (47) also provides a useful test for numerical codes for evaluating effective moduli, no matter if the component phases are isotropic or polycrystalline.

The authors are thankful to Michael Thorpe and Iwona Jasiuk for stimulating discussion on their current research (Thorpe & Jusiuk 1992) and are thankful to Gilles Francfort, Roderick Lakes and the referee for their helpful comments. A.V.C. and G.W.M. gratefully acknowledge the Packard Foundation, the Sloan Foundation, the Air Force Office of Scientific Research, through grant 90-0090, and the Army Research office, through grant DAAL 03-89-K-0039. A.V.C. and K.A.L gratefully acknowledge the Air Force Office of Scientific Research, through grant 90-0268.

References

Allaire, G. & Kohn, R. V. 1991 Explicit optimal bounds on the elastic energy of a two-phase composite in two-space dimensions. *Q. appl. Math.* (In the press.)

Atkin, R. J. & Fox, N. 1990 *An introduction to the theory of elasticity*. London: Longman.

Avellaneda, M. & Milton, G. W. 1989 Optimal bounds on the effective bulk modulus of polycrystals. *SIAM J. appl. Math.* **49**, 824–837.

Ball, J., Currie, J. C. & Olver, P. J. 1981 Null-Lagrangians, weak continuity, and variational problems of arbitrary order. *J. funct. Analysis* **41**, 4989–5003.

Cherkaev, A. V. & Gibiansky, L. V. 1991 Coupled estimates for the bulk and shear moduli of a two-dimensional isotropic elastic composites. *J. Mech. Phys. Solids.* (Submitted.)

Cherkaev, A. V., Gibiansky, L. V., Milton, G. W. & Rudelson, M. 1992 The range of moduli of isotropic polycrystals assembled from orthotropic monocrystals in planar elasticity. (In preparation.)

Day, A. R., Snyder, K. A., Garboczi, E. J. & Thorpe, M. F. 1991 The elastic moduli of a sheet containing circular holes. *J. Mech. Phys. Solids.* (In the press.)

Dundurs, J. 1967 Effect of elastic constants on stress in a composite under plane deformation. *J. Composite Mater.* **1**, 310–322.

Dykhne, A. M. 1970 Conductivity of a two-dimensional two-phase system. *Zh. Eksp. Teor. Fiz.* **59**, 110–115. (*Soviet Phys. JETP* **32** (1971), 63–65.)

Francfort, G. & Murat, F. 1986 Homogenization and optimal bounds in linear elasticity. *Arch. ration. Mech. Analysis* **94**, 307–334.

Francfort, G. & Tartar, L. 1991 Effective behavior of a mixture of isotropic materials with identical shear moduli. In *Mathematical problems in mechanics. C. r. Acad. Sci., Paris* I **312**, 301–307.

Gibiansky, L. & Cherkaev, A. 1987 Microstructures of composites of extremal rigidity and exact estimates of stored energy density. Preprint 1115, A. F. Ioffe Physico-Technical Institute, Academy of Sciences of U.S.S.R. (In Russian.)

Hashin, Z. & Shtrikman, S. 1963 A variational approach to the theory of the elastic behavior of multiphase materials. *J. Mech. Phys. Solids* **11**, 127–140.

Hill, R. 1952 The elastic behavior of a crystalline aggregate. *Proc. phys. Soc. Lond.* A **65**, 349–354.

Hill, R. 1964 Theory of mechanical properties of fibre-strengthened materials. I. Elastic behavior. *J. Mech. Phys. Solids* **12**, 199–212.

Keller, J. B. 1964 A theorem on the conductivity of a composite medium. *J. math. Phys.* **5**, 548–549.

Kohn, R. V. & Strang, G. 1986 Optimal design and relaxation of variational problems. *Commun. pure appl. Math.* **39**, 113–137, 139–182, 353–377.

Lakes, R. 1991 Deformation mechanisms in negative Poisson's ratio materials: structural aspects. *J. mater. Sci.* **26**, 2287.

Lipton, R. 1988 On the effective elasticity of a two-dimensional homogenized incompressible elastic composite. *Proc. R. Soc. Edinb.* A **110**, 45–61.

Lurie, K. A. & Cherkaev, A. V. 1984*a* G-closure of some particular sets of admissible material characteristics for the problem of bending of thin plates. *J. Opt. theor. Appl.* **42**, 305–316; also report 214 (1981), Danish Center for Applied Mathematics and Mechanics, Technical University of Denmark.

Lurie, K. A. & Cherkaev, A. V. 1984*b* Exact estimates of conductivity of mixtures composed of two isotropic media taken in prescribed proportion. *Proc. R. Soc. Edinb.* A **99**, 71–87; also preprint 783 (1982), A. F. Ioffe Physico-Technical Institute, Academy of Sciences of U.S.S.R. (In Russian.)

Lurie, K. A. & Cherkaev, A. V. 1986*a* Exact estimates of conductivity of a binary mixture of isotropic compounds. *Proc. R. Soc. Edinb.* A **104**, 21–38; also preprint 894 (1986), A. F. Ioffe Physico-Technical Institute, Academy of Sciences of U.S.S.R.

Lurie, K. A. & Cherkaev, A. V. 1986*b* The effective characteristics of composite materials and optimal design of construction. *Adv. Mech.* (*Poland*) **9**(2), 3–81. (In Russian.)

Michell, J. H. 1899 On the direct determination of stress in an elastic solid, with application to the theory of plates. *Proc. Lond. math. Soc.* **31**, 100–125.

Milton, G. W. 1988 Classical Hall effect in two-dimensional composites: a characterization of the set of realizable effective conductivity tensors. *Phys. Rev.* B **38**, 11296–11303.

Milton, G. W. 1990 On characterizing the set of possible effective tensors of composites: the variational method and the translation method. *Communs pure appl. Math.* **43**, 63–125.

Milton, G. W. 1991 A brief review of the translation method for bounding effective elastic tensors of composites. In *Continuum models and discrete systems* (ed. G. A. Maugin), vol. 1, pp. 60–74. Interaction of Mechanics and Mathematics Series. Essex: Longman.

Milton, G. W. 1992 Composite materials with Poisson's ratio close to -1. *J. Mech. Phys. Solids* **40**, 1105–1137.

Murat, F. & Tartar, L. 1985 Calcul des variations et homogénéisation. In *Les méthodes de l'homogénéisation: théorie et applications en physique*. Coll. de la Dir. des Etudes et Recherches de Electricité. de France, pp. 319–370. Paris: Eyrolles.

Stroud, D. & Bergman, D. J. 1984 New exact results for the Hall coefficient and magnetoresistance of inhomogeneous two-dimensional metals. *Phys. Rev.* B **30**, 447–449.

Tartar, L. 1985 Estimations fines des coefficients homogénéisés. In *Ennio de Giorgi Colloquium* (ed. P. Kree), vol. 125, pp. 168–187. Pitman Research Notes in Mathematics. London: Pitman Press.

Tartar, L. 1979 Compensated compactness and applications to partial differential equations. In *Nonlinear analysis and mechanics, Heriott-Watt Symp. VI* (ed. R. J. Knops). London: Pitman Press.

Thorpe, M. F. & Jasiuk, I. 1992 Discussion of some exact results in the theory of elasticity for two-dimensional composites. *Proc. R. Soc. Lond.* A **438**, 531–544. (Following paper.)

Timoshenko, S. P. 1959 *Theory of plates and shells*. New York: McGraw-Hill

Received 23 December 1991; revised 7 April 1992; accepted 28 February 1992